

IMPLEMENTASI METODE PAIRWISE COMPARISON PADA UJI KINERJA VARIAN METODE KECERDASAN BUATAN PADA PENYELESAIAN MASALAH TSP

Nama Mahasiswa : MUHAMMAD IBRAHIM OSWALDO
NRP : 5110 100 081
Jurusan : TEKNIK INFORMATIKA FTIF-ITS
Dosen Pembimbing : AHMAD SAIKHU, S.Si., MT.
BILQIS AMALIAH, S.Kom, M.Kom.

Abstrak

Dalam sebuah kasus pengambilan keputusan sering memerlukan perbandingan dari beberapa kriteria yang berbeda. Pairwise Comparison adalah metode perbandingan berpasangan yang dapat digunakan untuk memperoleh kecenderungan terkait dari setiap kriteria yang dibandingkan. Studi kasus yang diangkat adalah permasalahan TSP dengan permasalahan utama adalah pemilihan metode yang tepat untuk penyelesaian masalah TSP.

Nilai pairwise comparison matrix didapatkan berdasarkan tabel derajat kepentingan dari kriteria yang dibandingkan. Jika pairwise comparison matrix dapat diterima kelayakannya, dari proses normalisasi dan nilai bobot vektor yang didapat, kita dapat mengetahui nilai akhir dari setiap metode yang dibandingkan. Nilai akhir tertinggi yang dimiliki sebuah metode, menjadikan metode tersebut metode terbaik yang terpilih.

Dalam pembangunan model ini, penulis akan menggunakan matlab. Dari hasil uji coba model yang dibangun dapat disimpulkan bahwa algoritma basic ant colony optimization lebih baik dari basic genetic algorithm dalam penyelesaian masalah TSP.

Kata kunci: Basic Ant Colony Optimization (ACO), Basic Genetic Algorithm (GA), Kecerdasan Buatan, Pairwise Comparison Matrix, Travelling Salesman Problem (TSP).

PAIRWISE COMPARISON METHOD IMPLEMENTATION TO TEST PERFORMANCE OF VARIANTS ARTIFICIAL INTELLIGENCE METHOD WHICH SOLVING TSP PROBLEM

Name : MUHAMMAD IBRAHIM OSWALDO
NRP : 5110 100 081
Departement : INFORMATICS ENGINEERING
FTIF ITS
Supervisor : AHMAD SAIKHU, S.Si., MT.
BILQIS AMALIAH, S.Kom., M.Kom.

Abstract

In a case of decision making often requires the comparison of several different criterion. Pairwise Comparison is a method of paired comparisons that can be used to obtain the related tendency of each criterion being compared. The case study is the TSP problem which main problem is the selection of appropriate methods for solving the TSP.

Pairwise comparison matrix values obtained based on the degree of importance of the criterion table are compared. If the pairwise comparison matrix is acceptable feasibility, of the normalization process and the values of the weight vector is obtained, we can determine the final value of each method are compared. The best method is the method which has the highest end of the value.

In the construction of this model, authors will use matlab. From the test results that the model is built it can be concluded that the basic ant colony optimization algorithm is better than the basic genetic algorithm in solving TSP

Keywords: Basic Ant Colony Optimization (ACO), Basic Genetic Algorithm (GA), Kecerdasan Buatan, Pairwise Comparison Matrix, Travelling Salesman Problem (TSP).

DAFTAR KODE SUMBER

Kode Sumber 4.1-1 Kode Sumber Pembentukan Matriks A	63
Kode Sumber 4.1-2 Kode Sumber Proses Perhitungan RI....	63
Kode Sumber 4.1-3 Kode Sumber Pembentukan Normalisasi Matriks A	64
Kode Sumber 4.1-4 Kode Sumber Perhitungan Bobot Vektor Matriks A	64
Kode Sumber 4.1-5 Kode Sumber Perhitungan Nilai CI	64
Kode Sumber 4.1-6 Kode Sumber Evaluasi Kelayakan dari Matriks A	65
Kode Sumber 4.1-7 Kode Sumber Proses Pembentukan Matriks Turunan	65
Kode Sumber 4.1-8 Kode Sumber Pembentukan Normalisasi dari Matriks Turunan.....	66
Kode Sumber 4.1-9 Kode Sumber Perhitungan Bobot Matriks Turunan.....	67
Kode Sumber 4.1-10 Kode Sumber Evaluasi nilai GA dan ACO	67
Kode Sumber 4.2-1 Kode Sumber Implementasi Euclidean	68
Kode Sumber 4.3-1 Kode Sumber Inisialisasi Data Metode GA	69
Kode Sumber 4.3-2 Kode Sumber Pembangkitan Generasi Pertama	70
Kode Sumber 4.3-3 Kode Sumber Penentuan Titik Operator	70
Kode Sumber 4.3-4 Kode Sumber Proses Seleksi Parent	71
Kode Sumber 4.3-5 Kode Sumber Proses Penukaran Posisi Titik Operator.....	71
Kode Sumber 4.3-6 Kode Sumber Proses Pembentukan List L	72
Kode Sumber 4.3-7 Kode Sumber Proses Pembentukan List L'	73
Kode Sumber 4.3-8 Kode Sumber Proses Mencari Irisan List L dengan Child	73

Kode Sumber 4.3-9 Kode Sumber Proses Pembentukan Child	74
Kode Sumber 4.3-10 Kode Sumber Proses Mutasi	75
Kode Sumber 4.3-11 Kode Sumber Proses Evaluasi Fitness.	76
Kode Sumber 4.3-12 Kode Sumber Proses Elitisme	77
Kode Sumber 4.4-1 Kode Sumber Inisialisasi Data Metode ACO	78
Kode Sumber 4.4-2 Kode Sumber Proses Penempatan Semut	78
Kode Sumber 4.4-3 Kode Sumber Proses Penentuan Jalur Semut	79
Kode Sumber 4.4-4 Kode Sumber Proses Perhitungan Cost	80
Kode Sumber 4.4-5 Kode Sumber Proses Pembaharuan <i>Pheromones</i>	80
Kode Sumber 4.4-6 Kode Sumber Proses Penentuan Jalur Terpendek	81

BAB II

TINJAUAN PUSTAKA

Bab ini akan membahas tentang teori dasar yang menunjang penyusunan Tugas Akhir mengenai *Travelling Salesman Problem* (TSP), *Pairwise Comparison Matrix*, *Basic Genetic Algorithm*, *Basic Ant Colony Optimization* serta struktur penyelesaian TSP menggunakan *Basic Genetic Algorithm* dan *Basic Ant Colony Optimization* dalam pengerjaan Tugas Akhir.

2.1. Pairwise Comparison Matrix

Pairwise Comparison Matrix adalah metode perbandingan berpasangan yang digunakan dalam studi ilmiah. *Pairwise Comparison Matrix* biasanya mengacu pada setiap proses membandingkan setiap *varians* berpasangan untuk menilai yang mana dari setiap *varians* yang memiliki performa lebih baik. Pertama analisa hasil kinerja dari dua metode yang diimplementasikan pada persoalan TSP. Kemudian buat sebuah *comparison matrix* A yang berisi nilai positif integer setiap *varians*. *Comparison matrix* A ditunjukan pada Persamaan 1.

$$A = \begin{pmatrix} 1 & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & 1 & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & 1 & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & 1 \end{pmatrix} \quad (1)$$

Dari *matrix* tersebut dapat ditentukan bobot vektor UA . Setelah mendapatkan bobot vektor dapat dihitung nilai *consistency ratio matrix* untuk menentukan keseimbangan *matrix* dapat diterima atau tidak. Untuk mendapatkan nilai

consistency ratio ditunjukkan pada Persamaan 2 dan Persamaan 3 [5].

$$CI = \frac{n_{max}(A) - n}{n - 1} \quad (2)$$

$$CR = \frac{CI}{RI} \quad (3)$$

n merupakan kriteria yang ingin dibandingkan. $n_{max}(A)$ didapat setelah kita lakukan normalisasi dari matriks A . Sementara CI adalah nilai *consistency index* dan RI adalah nilai *acak consistency index*. Untuk nilai RI sudah ditetapkan seperti yang ditunjukkan pada Tabel 2.1-1.

Tabel 2.1-1 Koefisien Nilai RI

n	1	2	3	4	5	6	7	8	9	10
RI	0	0	0.58	0.9	1.12	1.24	1.32	1.41	1.46	1.49

Jika keseimbangan dari matriks A dapat diterima selanjutnya adalah membentuk matriks baru yang disebut matriks turunan. Jumlah matriks turunan yang kita buat sejumlah kriteria yang dibandingkan. Dengan proses yang sama kita mendapatkan bobot vektor dari masing-masing matriks turunan.

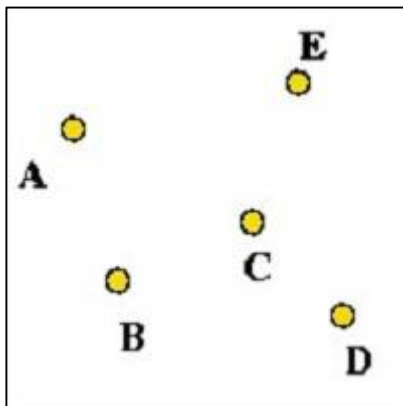
Terakhir analisa nilai setiap *varians* dari dua metode yang telah diimplementasikan pada persoalan TSP. Setiap kemenangan dari 2 metode yang dibandingkan akan mendapatkan nilai P . Nilai P adalah nilai akhir keseluruhan dari dua metode yang dibandingkan. Nilai P didapatkan berdasarkan penjumlahan total setiap bobot vektor matriks turunan dan matriks A .

2.2. Travelling Salesman Problem

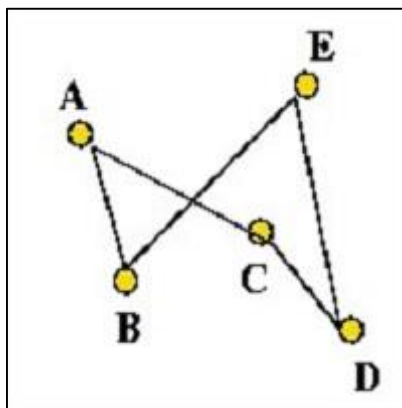
Permasalahan matematika tentang *Travelling Salesman Problem* pertama kali dikemukakan pada tahun 1800 oleh ilmuwan matematika Irlandia William Rowan Hamilton dan ilmuwan matematika Inggris Thomas Penyngton. Kode program computer yang dibuat untuk menyelesaikan persoalan TSP telah berkembang semakin baik dari tahun ke tahun. Tanda yang paling mencolok dari perkembangan metode untuk menyelesaikan persoalan TSP adalah bertambahnya jumlah *nodes* yang dapat diselesaikan, mulai dari solusi persoalan 49 kota yang dikembangkan oleh Dantzig, Fulkerson, dan Johnson pada tahun 1954 sampai kepada solusi persoalan 24.978 kota pada tahun 2004.

Travelling Salesman Problem termasuk permasalahan *non-deterministic polynomial time* paling terkenal, yang berarti bahwa tidak ada algoritma yang tepat untuk menyelesaikannya dalam waktu *polynomial*. Waktu tersingkat yang diharapkan untuk mendapatkan solusi optimal adalah eksponensial. TSP didefinisikan sebagai masalah permutasi dengan tujuan menemukan jalan terpendek. TSP dapat dimodelkan sebagai grafik berbobot tidak berarah, sehingga kota itu adalah *nodes*, jalan adalah *edges*, dan panjang jarak adalah panjang *edges*. Ini adalah masalah mendapatkan hasil yang paling minimal ketika seorang *salesman* memulai dan berakhir di suatu *nodes* yang telah ditentukan, setelah melalui setiap *nodes* lain tepat satu kali

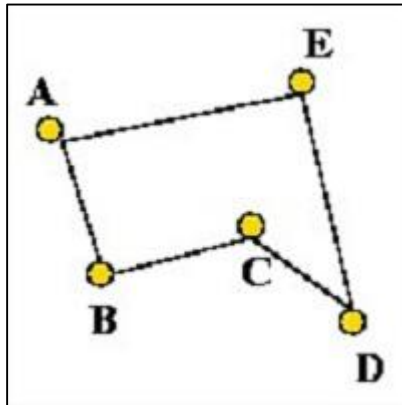
Untuk memahami bagaimana gambaran umum permasalahan TSP ini dan bagaimana hasil penyelesaiannya dapat dilihat pada Gambar 2.2-1 untuk ilustrasi letak persebaran kota, pada Gambar 2.2-2 untuk ilustrasi hasil penyelesaian yang kurang efektif dan pada Gambar 2.2-3 untuk ilustrasi hasil penyelesaian yang efektif.



**Gambar 2.2-2 Ilustrasi Letak
Persebaran *Nodes***



**Gambar 2.2-1 Ilustrasi
Penyelesaian Kurang Efektif**



**Gambar 2.2-3 Ilustrasi
Penyelesaian Yang Efektif**

Solusi langsung dari TSP adalah dengan menggunakan metode *Brute Force*, yaitu dengan mencoba semua kemungkinan yang ada (permutasi antara setiap kota). Metode ini akan menghasilkan hasil yang optimal, namun metode ini memiliki kompleksitas yang sangat tinggi yaitu ($O(n!)$), dimana n adalah jumlah kota yang ingin ditempuh. Metode penyelesaian TSP dibagi menjadi dua bagian, yaitu metode eksak dan metode heuristik. Metode eksak digunakan jika jumlah kota sedikit, karena metode eksak bekerja lambat. Jika jumlah kota banyak, lebih baik menggunakan metode heuristik yang bekerja secara cepat. Metode heuristik memiliki kelemahan yaitu hasil yang didapat belum dipastikan optimal, namun biasanya tetap mendekati optimal.

Telah disebutkan dalam batasan masalah bahwa data uji coba yang dipakai merupakan *fully connected graph*, sehingga jika pada persoalan TSP diberikan sejumlah n nodes, maka *graph* direpresentasikan dengan *adjacency matrix* berukuran $n \times n$. Jarak Euclidean d antara dua kota

dengan kordinat $(x1, y1)$ dan $(x2, y2)$ dapat dihitung dengan Persamaan 4.

$$d = \sqrt{(|x1 - x2|)^2 + (|y1 - y2|)^2} \quad (4)$$

2.3. Optimasi

Optimasi adalah suatu proses untuk mencari solusi optimal dari suatu permasalahan yang memiliki nilai tujuan maksimal atau minimal dengan tidak melanggar batasan-batasan yang diberikan. Terdapat bermacam-macam teknik optimasi seperti *Linear Programming* (LP), *Non Linear Programming* (NLP), *Integer Programming* (IP), *Goal Programming* (GP), *Basic Genetic Algorithm* (GA), *Basic Ant Colony Optimization* (ACO), *Particle Swarm Optimization* (PSO), dan lain sebagainya. Khusus pada Tugas Akhir ini akan menggunakan *Basic Genetic Algorithm* dan *Basic Ant Colont Optimization* yang akan digunakan dalam optimasi *Travelling Salesman Problem* (TSP).

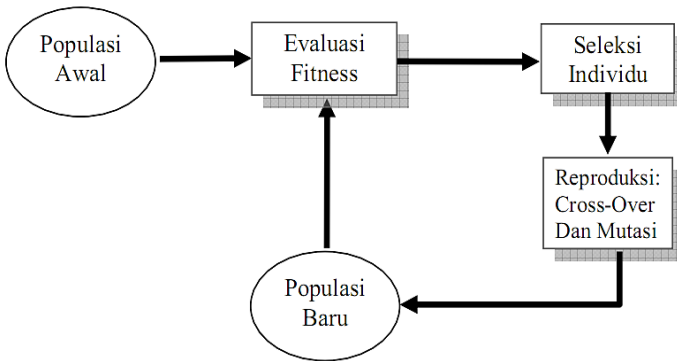
2.3.1. Basic Genetic Algorithm

Genetic Algorithm atau algoritma genetika (GA) masuk dalam kelompok *Evolutionary Algorithm*. GA didasarkan pada prinsip-prinsip genetika dan seleksi alam. Elemen-elemen dasar dari genetika alam adalah reproduksi, *crossover* atau kawin silang, dan mutasi. Elemen-elemen ini yang dipakai dalam prosedur GA. Algoritma ini banyak dipakai dalam penyelesaian masalah kombinatorial seperti TSP. GA termasuk pelopor dalam pendekatan Metaheuristik. Banyak algoritma yang belakangan muncul mengadopsi beberapa langkah dari GA. Dalam *evolution-based approach* biasanya akan dibangkitkan sejumlah populasi yang dalam masaah optimasi menjadi solusi awal. Dengan prosedur tertentu seperti mutasi, seleksi, dan *crossover* akhirnya

didapatkan solusi akhir dari problem optimasi yang dihadapi. GA termasuk temuan penting dalam bidang optimasi, dimana suatu algoritma diciptakan dengan meniru mekanisme makhluk hidup. Dalam GA prosedur pencarian hanya didasarkan pada nilai fungsi tujuan, tidak ada pemakaian Gradient atau teknik kalkulus

2.3.1.1. Konsep Dasar GA

Gambaran umum bagaimana jalannya algoritma genetika dapat dilihat pada Gambar 2.3-1.



Gambar 2.3-1 Siklus GA

Penjelasan bagian-bagian serta operasinya akan dijelaskan pada uraian berikut:

1. Bangkitkan populasi awal

Bangkitkan populasi awal atau kromosom-kromosom awal secara acak sesuai ukuran populasi yang diinginkan. Evaluasi nilai setiap individu di dalam populasi awal ini dengan menggunakan fungsi *fitness*. Probabilitas kawin silang dan probabilitas mutasi.

2. Set iterasi $t = 1$.

3. Pilih individu terbaik untuk disalin sejumlah tertentu untuk mengganti individu lain (elitisme).
4. Lakukan seleksi kompetitif untuk memilih anggota populasi sebagai induk untuk dilakukan kawin silang.
5. Lakukan kawin silang antar induk yang terpilih.
6. Tentukan beberapa individu dalam populasi untuk mengalami proses mutasi.
7. Jika belum mencapai konvergensi set iterasi $t = t + 1$.
8. Kembali ke langkah 3.

Beberapa istilah yang akan dipakai dalam GA dijelaskan dalam subbab berikut ini.

2.3.1.2. Kromosom

Dalam GA, kromosom merupakan bagian penting dari algoritma. Satu kromosom atau individu mewakili satu vektor solusi. Vektor solusi terkadang bisa langsung diimplementasikan dalam GA tetapi terkadang bisa juga dilakukan *encoding* atau pengodean. Pengodean dilakukan untuk mewakili suatu nilai solusi dengan menggunakan bilangan biner. Ini tergantung pada permasalahan optimasi yang dihadapi. Dalam permasalahan optimasi fungsi, sering kali nilai kontinyus akan diwakili dengan bilangan biner. Dengan bilangan biner langkah-langkah kawin silang atau mutasi akan lebih banyak variasinya. Sebaliknya bisa juga dilakukan perhitungan tanpa melalui proses pengodean. Jadi solusi memang dalam bentuk kontinyus dan proses-proses seleksi, kawin silang, dan mutasi dilakukan dengan menggunakan bilangan kontinyus. Pada akhir algoritma GA, jika dilakukan pengodean maka akan dilakukan proses mengembalikan ke nilai kontinyus asalnya, yang sering disebut *decoding*. Dalam GA kita akan membangkitkan populasi sebagai kumpulan dari kromosom, dimana masing-masing kromosom mewakili suatu vektor solusi. Dengan dibangkitkannya populasi ini, maka akan tersedia banyak pilihan solusi.

2.3.1.3. Fitness

Fungsi *fitness* digunakan untuk mengukur tingkat kebaikan atau kesesuaian suatu solusi yang dicari. Fungsi *fitness* bisa berhubungan langsung dengan fungsi tujuan, atau bisa juga sedikit modifikasi terhadap fungsi tujuan. Sejumlah solusi yang dibangkitkan dalam populasi akan dievaluasi menggunakan fungsi *fitness*. Fungsi *fitness* yang dipakai dalam pengerjaan Tugas Akhir ditunjukkan pada Persamaan 5.

$$F(x) = \frac{1}{1 + f(x)} \quad (5)$$

Dimana $f(X)$ adalah fungsi tujuan dari problem yang kita selesaikan. Untuk kasus minimasi, jika didapatkan $f(X)$ yang kecil maka nilai *fitness*-nya besar. Sebaliknya, untuk kasus maksimasi, fungsi *fitness*-nya bisa menggunakan nilai $f(X)$ sendiri, jadi $F(X) = f(X)$.

Setelah setiap solusi dievaluasi dengan fungsi *fitness*, perlu dilakukan proses seleksi terhadap kromosom. Proses seleksi dilakukan untuk memilih diantara kromosom anggota populasi ini, mana yang bisa menjadi *parent* (induk) atau melakukan identifikasi diantara populasi ini, kromosom yang akan menjadi anggota populasi berikutnya. Ada beberapa cara melakukan seleksi ini. Sebagian anggota populasi bisa dipilih untuk proses reproduksi. Cara yang umum digunakan adalah melalui *roulette wheel selection* atau roda lotere

2.3.1.4. Elitisme

Konsep *elitism* (elitisme) dalam GA berarti usaha mempertahankan individu-individu terbaik yang telah diperoleh pada suatu generasi ke dalam generasi selanjutnya. Sehingga individu-individu terbaik ini akan tetap muncul pada populasi berikutnya. Langkah ini dilakukan dalam berbagai cara. Misalnya, melalui penyalinan individu terbaik, atau dapat juga melalui kombinasi antara solusi-solusi turunan atau

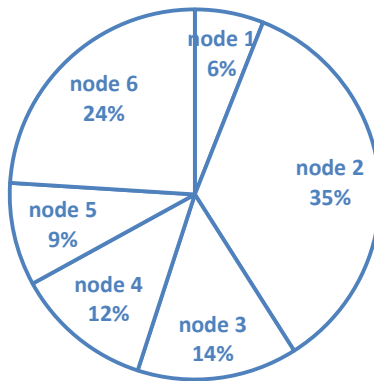
anak dengan induk. Terbukti bahwa penggunaan operator elitisme ini telah terbukti memiliki pengaruh yang sangat penting saat menggunakan GA untuk menyelesaikan persoalan optimasi dengan tujuan tunggal.

2.3.1.5. Seleksi

Biasanya GA melakukan seleksi menggunakan proses roda lotere. Untuk memahami proses seleksi dengan roda lotere kita lihat Gambar 2.3-2 Kita bayangkan mempunyai lingkaran lotere yang daerahnya dibagi-bagi sebanyak jumlah individu atau solusi yang dibangkitkan. Dalam contoh ini ada 6 individu. Dalam roda lotere setiap individu atau solusi yang dibangkitkan punya kesempatan untuk terpilih. Setiap area dalam lingkaran ini menunjukkan peluang setiap solusi untuk dipilih. Untuk setiap individu atau solusi dibangkitkan bilangan acak r . Misalkan untuk individu 1, dibangkitkan bilangan acak (atau dalam contoh ini biangan acak dikalikan dengan total *fitness* (tf)) berada di area mana dari lingkaran lotere tersebut. Misalkan nilai $r \times tf$ lebih kecil dari akumulasi *fitness* 1 + *fitness* 2, maka individu 2 yang akan dipilih untuk menggantikan individu 1.

Setiap kali roda diputar sampai berhenti, jarum penunjuk akan tertuju pada individu tertentu. Individu dengan nilai *fitness* paling besar akan punya kesempatan terpilih paling besar untuk menjadi induk. Dalam contoh ini, jika roda diputar sebanyak jumlah individu (6), peluang individu 1 dengan nilai *fitness* 35 akan terpilih paling besar, dibanding individu dengan *fitness* yang lebih rendah. Dalam optimasi *fitness* ini mewakili nilai fungsi tujuan. Dalam kasus minimasi fungsi, semakin kecil nilai fungsi tujuan, semakin besar nilai *fitness*. Proses seleksi induk ini akan diulang sejumlah yang diinginkan.

Dari contoh ini, terlihat bahwa solusi ke-2 punya peluang paling tinggi untuk diseleksi menjadi induk karena kromosom ke-2 ini mempunyai nilai *fitness* yang paling besar.



Gambar 2.3-2 Roda Lotere

2.3.1.6. Crossover atau Kawin Silang

Istilah *crossover* juga sering disebut kawin silang. Kawin silang dilakukan untuk mendapatkan kombinasi yang lebih baik antara satu individu dengan individu yang lain dalam satu populasi. Ada bermacam-macam teknik yang bisa digunakan dalam GA.

Kombinasi linier parameter dalam kawin silang adalah probabilitas kawin-silang. Jika parameter ini bernilai kecil, maka hanya akan sedikit kromosom yang akan mengalami kawin silang. Jika nilai ini membesar, maka akan semakin besar kromosom yang akan mengalami kawin silang. Misalkan nilai parameter ini adalah 0.5, maka akan ada sekitar 50% atau separuh populasi yang akan mengalami kawin silang.

2.3.1.7. Mutasi

Mutasi memungkinkan munculnya individu-individu baru yang bukan berasal dari hasil kawin silang. Mutasi dimaksudkan untuk memunculkan individu baru yang berbeda

sama sekali dengan individu yang sudah ada. Dalam konteks optimasi memungkinkan munculnya solusi baru untuk bisa keluar dari optimum lokal. Mutasi mengacu pada perubahan urutan atau penggantian elemen dari vektor solusi (pada problem TSP) dan pemunculan nilai baru (optimasi fungsi). Elemen tersebut juga dipilih secara acak.

Parameter penting dalam mutasi adalah probabilitas mutasi. Probabilitas ini akan menentukan kromosom mana yang akan mengalami perubahan gen. Semakin besar nilai probabilitas mutasi, semakin banyak kromosom dalam populasi yang akan mengalami mutasi. Misalkan nilai probabilitas mutasi adalah 0.01, maka akan ada sekitar 1% dari seluruh kromosom dalam populasi yang akan mengalami mutasi.

2.3.2. Basic Ant Colony Optimization

Dalam dua dekade terakhir ini, banyak penelitian yang berhubungan dengan komputasi menggunakan perilaku alam sebagai dasar penelitiannya. Salah satunya adalah *Ant Colony*. Penelitian mengenai *Ant Colony* telah menyumbangkan pengetahuan yang sangat besar terhadap bidang *bioinformation*, yang merupakan observasi bagaimana suatu kejadian yang ada di alam dapat digunakan untuk mengatasi permasalahan yang ada pada dunia industri. *Ant Colony Optimization* (ACO) termasuk dalam kelompok *Swarm Intelligence*, yang merupakan salah satu jenis pengembangan paradigma yang digunakan untuk menyelesaikan masalah optimasi dimana inspirasi yang digunakan untuk memecahkan masalah tersebut berasal dari perilaku kumpulan atau *swarm* (kawanan) serangga.

ACO biasanya digunakan untuk menyelesaikan *discrete optimization problems* dan persoalan yang kompleks dimana terdapat banyak variabel. Hasil yang diperoleh menggunakan ACO, walaupun tidak optimal namun mendekati optimal. ACO sudah diterapkan pada berbagai

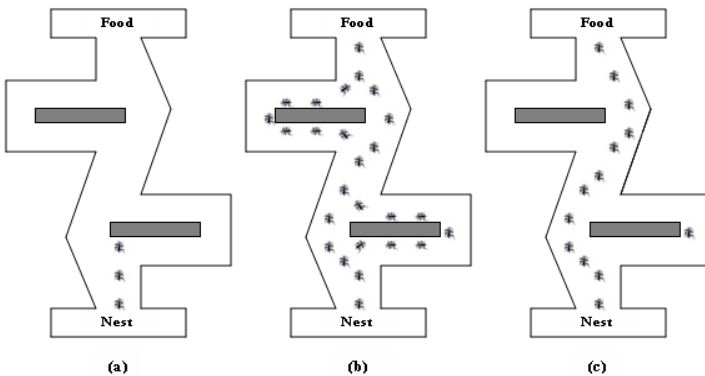
masalah seperti TSP, penjadwalan proyek dengan sumber daya terbatas, penggalian data, penjadwalan pekerjaan, dan beberapa masalah kombinatorial yang lain. Teknik *Ant-based* pertama kali digunakan oleh Dorigo et. Al Dorigo et al [1996] dengan menggunakan ACO untuk menyelesaikan *Travelling Salesman Problem* (TSP). Dalam ACO, setiap semut dalam kawanan yang berjalan akan meninggalkan *pheromones* (semacam zat kimia) pada jalur yang dilaluinya. *Pheromones* ini menjadi semacam sinyal bagi sesama semut. Jalur yang pendek akan menyisakan sinyal yang lebih kuat. Semut berikutnya, pada saat memutuskan jalur mana yang harus dipilih, biasanya akan cenderung memilih untuk mengikuti jalur dengan sinyal yang paling kuat, sehingga jalur terpendek akan ditemui karena lebih banyak semut yang akan melewati jalur tersebut. Semakin banyak semut yang lewat suatu jalur, semakin kuat sinyal pada jalur itu. Hal ini dapat dilihat pada Gambar 2.2, dimana ditunjukkan bagaimana jalur yang dilalui oleh semut pada saat mencari makan dari sarang sampai sumber makanan.

2.3.2.1. Konsep Dasar ACO

Konsep ACO sendiri terinspirasi dari pengamatan terhadap tingkah laku semut. Secara sederhana, perilaku semut dapat digambarkan sebagai berikut. Semut merupakan serangga sosial yang hidup dalam koloni-koloni dan berperilaku berdasarkan kepentingan koloni. Salah satu hal yang menarik dari perilaku semut adalah kemampuan dalam menemukan jarak terpendek antara sarang mereka dan sumber makanan. Misalkan ada segerombolan semut yang berada di depan harus memilih lintasan tertentu untuk dilewati. Pada saat semut pertama berjalan, semut tersebut meninggalkan hormone *pheromones* yang dapat dicium oleh semut berikutnya, sehingga semut-semut berikutnya tahu apakah tempat tersebut sudah dilewati atau belum. Dengan demikian, setiap semut yang berjalan akan meninggalkan *pheromones*

pada jalur yang dilalui. ACO meniru perilaku koloni semut dalam mencari sumber makanan dan kembali ke sarangnya yang ternyata secara alami mencari jalur terpendek.

Semut yang melewati lintasan yang pendek akan meninggalkan aroma *pheromones* yang lebih tajam daripada yang menempuh jalur lintasan yang lebih panjang. Hal ini terjadi karena *pheromones* yang ditinggalkan dapat menguap. Saat semut-semut yang di belakang mengikuti semut-semut yang di depannya, semut-semut tersebut akan memilih lintasan berdasarkan kekuatan aroma *pheromones* dan jarak lintasan. Semakin banyak semut yang menempuh suatu lintasan tertentu, maka aroma *pheromones* pada lintasan tersebut akan semakin kuat sehingga semut-semut berikutnya akan mengikuti lintasan yang sama, sehingga jalur terpendek akan ditemukan karena lebih banyak semut yang akan melewati jalur tersebut. Gambar 2.3-3 menunjukkan bagaimana koloni semut berjalan dari sarang menuju sumber makanan.



Gambar 2.3-3 Ilustrasi Koloni Semut Mencari Rute Terpendek

Pada awalnya semut akan terdistribusi merata melalui dua lintasan. Setelah beberapa saat semut-semut mulai memilih lintasan terpendek karena *pheromones* yang ditinggalkan semut akan mengumpul di lintasan yang lebih pendek.

Salah satu hal yang penting pada *ant-inspired technologies* adalah pada saat sistem telah menemukan solusi yang optimal, ACO akan beradaptasi dengan cepat terhadap perubahan yang terjadi di sekitar. Adaptasi ini didasarkan pada *pheromones* yang merupakan dasar dari *ant-system*. *Pheromones* yang lebih kuat akan dimiliki oleh solusi dengan jalur yang lebih optimal pada akhir suatu algoritma.

Ant Colony Optimization (ACO) didasarkan pada perilaku kerja sama dari koloni semut yang dapat menemukan lintasan terpendek dari sarangnya menuju sumber makanan. Proses ACO dapat dijelaskan dengan membuat grafik multilayer dari permasalahan optimasi yang dihadapi. Seperti ditunjukkan dalam Gambar 2.3-4 dimana jumlah layer sama dengan jumlah variabel dari permasalahan yang dihadapi dan jumlah *nodes* (simpul) dalam suatu layer tertentu sama dengan jumlah nilai diskret yang diijinkan untuk variabel yang berkaitan. Sehingga setiap simpul berkaitan dengan satu nilai diskret yang dimungkinkan untuk suatu variabel. Gambar 2.3-4 menunjukkan suatu problem dengan 5 variabel dan 4 kemungkinan nilai untuk setiap variabel.

Proses dalam ACO bisa dijelaskan sebagai berikut. Misalkan ada N semut dalam suatu koloni. Semut-semut itu akan memulai gerakan dari sarang mereka menuju tujuan akhir melalui beberapa layer, dari layer pertama menuju layer terakhir dan berakhir pada simpul tujuan di akhir setiap siklus atau iterasi. Setiap semut hanya bisa memilih satu simpul dalam setiap layer dengan prosedur tertentu untuk memilihnya seperti dalam Persamaan 6. Simpul-simpul yang dipilih sepanjang lintasan ini merupakan kandidat solusi. Misalnya

suatu lintasan telah dilalui dengan diperlihatkan sebagai gari tebal pada gambar. Maka solusi yang didapat dari lintasan ini adalah $(x_{12}, x_{22}, x_{33}, x_{44}, x_{54})$. Sekali lintasan terpilih, semut-semut akan menyimpan sejumlah *pheromones* pada lintasan tersebut menggunakan rumus.

Kalau semua semut sudah menyelesaikan lintasanya, jumlah *pheromones* pada lintasan terbaik secara global akan diperbarui dengan rumus. Lintasan global terbaik artinya terbaik diantara semua semut. Pada awal proses yaitu pada iterasi 1, semua ruas dari simpul awal akan diberi jumlah *pheromones* yang sama. Sehingga pada iterasi 1, semua semut akan mulai dari simpul awal dan berakhir pada simpul tujuan dengan cara memilih simpul-simpul antara secara acak. Proses optimasi berakhir jika jumlah iterasi maksimum sudah tercapai atau tidak ada lagi solusi yang lebih baik yang bisa didapat dalam beberapa iterasi yang berturutan. Vektor solusi didapatkan dari simpul-simpul pada lintasan yang mempunyai jumlah optimal semua semut menempuh lintasan terbaik yang sama. Dalam hal minimasi fungsi lintasan terbaik memberikan nilai fungsi paling rendah.

2.3.2.2. Perilaku Semut

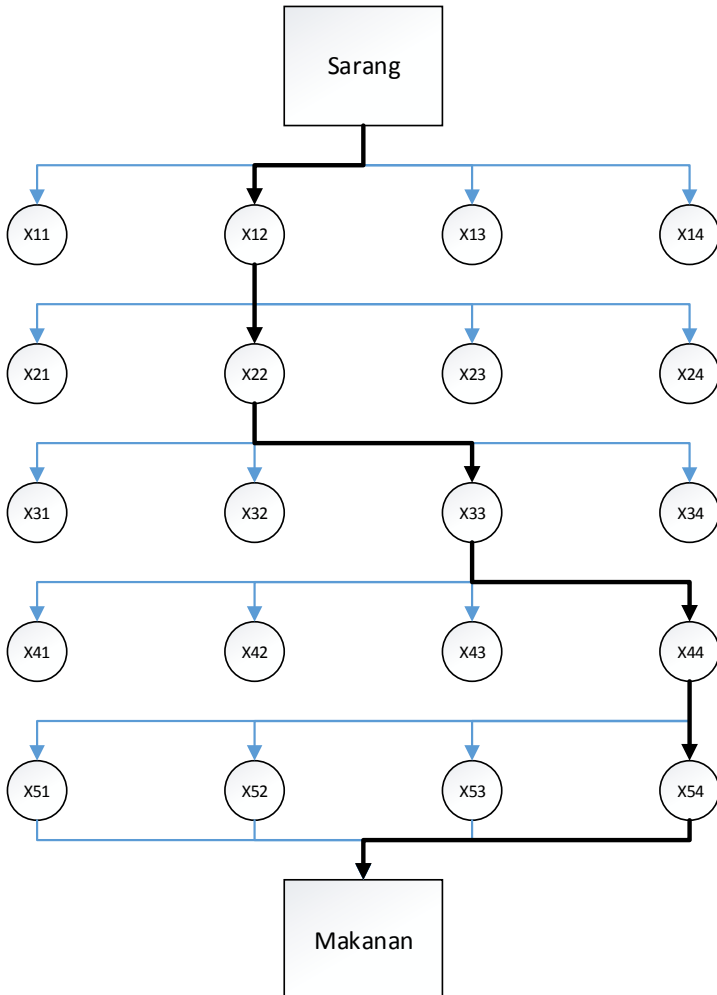
Seekor semut k pada simpul i akan memilih simpul j yang dituju pada simpul berikutnya dengan probabilitas seperti yang ditunjukkan pada Persamaan 6.

$$p_{i,j} = \frac{\tau_{i,j}^\alpha}{\sum_{j \in N_i^{(k)}} \tau_{i,j}^\alpha}, \text{ jika } j \in N_i^{(k)} \quad (6)$$

$$p_{i,j} = 0, \quad \text{jika } j \notin N_i^{(k)}$$

Dimana α menunjukkan derajat kepentingan *pheromones* dan $N_i^{(k)}$ adalah tetangga pilihan yang dipunyai semut k pada saat ia berada pada simpul i . Tetangga dari semut k pada simpul i akan mengandung semua simpul yang

bisa dituju yang tersambung secara langsung ke simpul i , kecuali simpul yang sudah dikunjungi sebelumnya.



**Gambar 2.3-4 Representasi Grafis Proses ACO dalam Bentuk
*Network Multi-Layer***

2.3.2.3. Penambahan dan Penguapan Pheromones

Seekor semut k ketika melewati ruas akan meninggalkan *pheromones*. Jumlah *pheromones* yang terdapat pada ruas i, j setelah dilewati semut k diberikan dengan Persamaan 7.

$$\tau_{ij} = \tau_{ij} + \Delta\tau^k \quad (7)$$

Dengan meningkatnya nilai *pheromones* pada *nodes* $i - j$, maka kemungkinan ruas ini akan dipilih lagi pada iterasi berikutnya semakin besar. Setelah jumlah *nodes* dilewati maka akan terjadi penguapan *pheromones* yang ditunjukkan dengan Persamaan 8.

$$\tau_{ij} = (1 - \rho)\tau_{ij}; \forall (i, j) \in A \quad (8)$$

Dimana $\rho \in (0,1]$ adalah parameter tingkat penguapan dan A menyatakan *nodes* yang sudah dilalui oleh semut k sebagai bagian dari lintasan dari sarang menuju makanan. Penurunan jumlah *pheromones* memungkinkan semut untuk mengeksplorasi lintasan yang berbeda selama proses pencarian. Ini juga akan menghilangkan kemungkinan memilih lintasan yang kurang bagus. Selain itu, ini juga membantu membatasi nilai maksimum yang dicapai oleh suatu lintasan *pheromones*. Jumlah *pheromones* yang ditambah pada $i - j$ oleh semut k ditunjukkan pada Persamaan 9.

$$\Delta\tau_{i,j}^{(k)} = \frac{Q}{L_k} \quad (9)$$

Dimana Q adalah konstanta dan L_k adalah lintasan terpendek yang dilalui semut k (dalam kasus TSP adalah lintasan yang dilalui oleh *salesman*). Nilai Q biasanya ditentukan oleh user. Atau bisa juga diimplementasikan dengan Persamaan 10.

$$\begin{aligned} \Delta\tau_{i,j}^{(k)} &= \frac{cf_{best}}{f_{worst}}, & \text{jika } (i, j) \in \text{lintasan terbaik global} \\ \Delta\tau_{i,j}^{(k)} &= 0, & \text{untuk yang lain} \end{aligned} \quad (10)$$

Dimana f_{best} adalah nilai terbaik dari fungsi tujuan dan f_{worst} nilai terjelek dari fungsi tujuan. Sedangkan c adalah konstanta untuk mengontrol skala pembaharuan semua *pheromones*. Semakin tinggi nilai c semakin banyak *pheromones* ditambahkan ke lintasan global terbaik dan semakin bagus kemampuan mengeksplorasi. Persamaan 10 dimaksudkan untuk memberikan jumlah *pheromones* lebih banyak pada lintasan terbaik secara keseluruhan.

2.3.2.4. Urutan Dasar ACO

Tahap-tahap algoritma ACO untuk menyelesaikan problem minimasi fungsi dapat dijelaskan dalam ringkasan berikut:

1. Asumsikan jumlah semut sebanyak N . Tentukan juga sejumlah nilai diskret sebanyak p yang akan digunakan untuk mencari nilai variabel yang ada pada permasalahan yang kita selesaikan. Nilai ini dinyatakan sebagai $x_i = x_{i1}, x_{i2}, \dots, x_{ip}, (i = 1, 2, \dots, n)$, dimana n adalah banyaknya variabel. Tentukan jumlah *pheromones* awal τ_{ij}^1 yang sama untuk semua ruas yang ada pada jaringan *multi-layer* seperti ditunjukkan dalam Gambar 2.4. Set iterasi, $t = 1$.
2. Hitung probabilitas (p_{ij}) untuk memilih *nodes* atau nilai diskret x_{ij} menggunakan Persamaan 11.

$$p_{ij} = \frac{\tau_{ij}}{\sum_{j \in N_i^{(k)}} \tau_{ij}}; i = 1, 2, \dots, n; j = 1, 2, \dots, p \quad (11)$$

Yang sama dengan Persamaan 6 dengan nilai $\alpha = 1$. Nilai α yang lebih besar bisa juga digunakan.

Kemudian *nodes* tertentu akan dipilih oleh semut k berdasarkan bilangan acak dalam jangkauan (0,1). Untuk itu kita perlu juga menentukan range probabilitas komulatif yang berkaitan dengan pilihan *nodes*. Jadi jika

- ada p kemungkinan nilai variabel, maka akan ada p pilihan jangkauan probabilitas.
3. Bangkitkan N bilangan acak r_1, r_2, \dots, r_N dalam range $(0,1)$, satu untuk setiap semut. Tentukan nilai diskret yang mewakili *nodes* untuk semut k untuk variabel i dengan menggunakan bilangan acak dari tahap 2. Ulangi proses tersebut untuk semua variabel $i = 1, 2, \dots, n$. Dan evaluasi nilai fungsi tujuan dengan cara memasukan nilai x_{ij} yang sudah dipilih untuk semua variabel $i = 1, 2, \dots, n$ oleh semut k , $k = 1, 2, \dots, N$: $f_k = f(X(k))$; $k = 1, 2, \dots, N$. Tentukan lintasan terbaik dan terburuk diantara N *nodes* atau lintasan yang sudah dipilih oleh semut-semut yang berbeda.
 4. Uji konvergensi dari proses. Dalam hal ini konvergensi bisa diartikan jika semua semut mengambil lintasan terbaik yang sama. Jika belum konvergen, koloni semut akan kembali ke sarang dan memulai pencarian makanan lagi [6].

2.4. Metode Heuristik

Metode Heuristik adalah sebuah pendekatan dalam penyelesaian masalah. Metode ini biasa digunakan ketika suatu algoritma atau metode konvensional cukup lambat atau bahkan tidak dapat mencari solusi yang diinginkan. Tujuan dari penggunaan metode heuristik adalah menghasilkan solusi yang cukup bagus dari suatu permasalahan, walaupun tidak dapat dipastikan solusi tersebut adalah solusi yang terbaik.

Metode Heuristik akan digunakan dalam GA untuk mencari solusi terbaik berupa individu/kromosom terbaik dari *solution space* yang ada, dimana yang menjadi solusi terbaik dalam permasalahan TSP adalah total jarak yang paling pendek.

BAB III

METODOLOGI PENELITIAN

Pada bab ini akan diuraikan mengenai desain dan perancangan sistem perangkat lunak agar dapat mencapai tujuan dari Tugas Akhir ini. Perangkat lunak yang dibuat pada Tugas Akhir ini berguna untuk mendapatkan metode kecerdasan buatan terbaik untuk penyelesaian masalah TSP berdasarkan tingkat kinerja *varians* dari dua metode yang dibandingkan. Perangkat lunak ini memiliki tiga bagian utama yaitu proses penyelesaian masalah TSP menggunakan *Basic Genetic Algorithm*, proses penyelesaian masalah TSP menggunakan *Basic Ant Colony Optimization*, dan proses membandingkan kinerja *varians* dari dua metode tersebut menggunakan *Pairwise Comparison Matrix*. Penjelasan pada bab ini akan dibagi menjadi dua bagian yaitu perancangan data, dan pembahasan metode kecerdasan buatan untuk menyelesaikan permasalahan TSP.

3.1. Perancangan Data

Pada bagian ini, akan dijelaskan perancangan data yang akan digunakan dalam proses penyelesaian masalah TSP dengan dua metode kecerdasan buatan yaitu GA dan ACO, agar bisa dibandingkan *varians* pada dua metode tersebut. Perancangan data akan dibagi menjadi 3 bagian yaitu *input* (data masukan), data proses, dan *output* (data keluaran).

3.1.1. Data Masukan

Data masukan pada TSP adalah data awal permasalahan. Data masukan ini terdiri dari 3 bagian yaitu data nomor *nodes* atau kota, data titik sumbu *x* kota, dan data titik sumbu *y* kota. Contoh data masukan dapat dilihat pada Tabel 3.1-1 sampai dengan Tabel 3.1-2. Untuk data masukan lainnya dapat dilihat pada Lampiran A 1 sampai dengan Lampiran A 7.

Tabel 3.1-1 Data Masukan *eil51* (Bagian 1)

Nomor Kota	Sumbu x	Sumbu y
1	37	52
2	49	49
5	40	30
6	21	47
7	17	63
8	31	62
9	52	33
10	51	21
11	42	41
12	31	32
13	5	25
14	12	42
15	36	16
16	52	41
17	27	23
18	17	33
19	13	13
20	57	58
21	62	42
22	42	57
23	16	57
24	8	52
25	7	38
26	27	68
27	30	48
28	43	67
29	58	48
30	58	27
31	37	69
32	38	46
33	46	10
34	61	33

Tabel 3.1-2 Data Masukan *eil51* (Bagian 2)

Nomor Kota	Sumbu x	Sumbu y
35	62	63
36	63	69
37	32	22
38	45	35
39	59	15
40	5	6
41	10	17
42	21	10
43	5	64
44	30	15
45	39	10
46	32	39
47	25	32
48	25	55
49	48	28
50	56	37
51	30	40

3.1.2. Data Proses

Pada bagian ini akan dijelaskan data-data yang digunakan dalam proses penyelesaian TSP menggunakan *Basic Genetic Algorithm* dan *Basic Ant Colony Optimization*.

3.1.2.1. Data Proses GA

Data proses untuk metode *Basic Genetic Algorithm* terdapat pada Tabel 3.1-3 sampai dengan Tabel 3.1-4.

Tabel 3.1-3 Data Proses Penyelesaian TSP dengan metode GA (Bagian 1)

No	Nama Data	Tipe Data	Keterangan
1	jumlah	<i>Integer</i>	Jumlah kota dalam TSP
2	jarak	<i>Double</i>	Matriks yang menyimpan data jarak antar kota yang berukuran jumlah x jumlah
3	parent	<i>Integer</i>	Matriks yang menyimpan setiap kromosom individu
4	parentPCO	<i>Double</i>	Berisi nilai acak yang dibangkitkan untuk setiap <i>parent</i> untuk perbandingan dengan probabilitas <i>crossover</i> yang telah ditentukan
5	individu	<i>Integer</i>	Matriks yang menyimpan seluruh <i>parent</i> dan <i>child</i>
6	populasi	<i>Integer</i>	Jumlah populasi pada GA
7	maksiterasi	<i>integer</i>	Jumlah perulangan yang ingin dilakukan
8	Pco	<i>Double</i>	Berisi nilai probabilitas <i>crossover</i>
9	Pcm	<i>Double</i>	Berisi nilai probabilitas mutasi
10	m1	<i>Integer</i>	Operator mutasi 1
11	m2	<i>integer</i>	Operator mutasi 2
12	c1	<i>Integer</i>	Operator <i>crossover</i> 1
13	c2	<i>Integer</i>	Operator <i>crossover</i> 2
14	Child	<i>Integer</i>	Matriks yang menyimpan kromosom hasil dari proses <i>crossover</i>
15	availCO	<i>Integer</i>	Berisi indeks <i>parent</i> yang akan mengalami <i>crossover</i>

**Tabel 3.1-4 Data Proses Penyelesaian TSP dengan metode GA
(Bagian 2)**

No	Nama Data	Tipe Data	Keterangan
16	iterCO	<i>Integer</i>	Berisi nilai perulangan berapa banyak <i>crossover</i> dilakukan
17	L	<i>Integer</i>	Matriks yang digunakan saat proses <i>crossover</i> untuk menyimpan nilai operator <i>crossover</i> sementara
18	Laksen	<i>Integer</i>	Matriks yang digunakan saat proses <i>crossover</i> untuk menyimpan nilai operator <i>crossover</i> yang beririsan dengan <i>child</i> sementara
19	childPM	<i>Double</i>	Berisi nilai acak yang dibangkitkan untuk setiap <i>child</i> untuk perbandingan dengan probabilitas mutasi yang telah ditentukan
20	availM	<i>Integer</i>	Berisi indeks <i>child</i> yang akan mengalami mutasi
21	Fitness	<i>Double</i>	Matriks yang menyimpan hasil dari proses penyelesaian TSP menggunakan GA

3.1.2.2. Data Proses ACO

Data proses untuk metode *Basic Ant colony Optimization* terdapat pada Tabel 3.1-5 sampai dengan Tabel 3.1-6.

**Tabel 3.1-5 Data Proses Penyelesaian TSP dengan metode ACO
(Bagian 1)**

No	Nama Data	Tipe Data	Keterangan
1	Jumlah	<i>Integer</i>	Jumlah kota dalam TSP
2	Distance	<i>Double</i>	Matriks yang menyimpan data jarak antar kota yang berukuran jumlah x jumlah
3	maksiterasi	<i>Integer</i>	Jumlah perulangan yang ingin dilakukan
4	M	<i>Integer</i>	Jumlah semut
5	Eva	<i>Double</i>	Berisi nilai koefisien dari evaporasi
6	Eli	<i>Double</i>	Berisi nilai koefisien dari eliminasi
7	Alpha	<i>Double</i>	Berisi nilai koefisien untuk urutan kemungkinan <i>next state</i>
8	Beta	<i>Double</i>	Berisi nilai koefisien untuk urutan <i>state</i> yang telah di lewati
9	Pheromones	<i>Double</i>	Matriks yang menyimpan nilai <i>pheromones</i> pada setiap jalur
10	T	<i>Double</i>	Matriks yang menampung nilai <i>pheromones</i> yang terupdate untuk perbandingan <i>next state</i>

**Tabel 3.1-6 Data Proses Penyelesaian TSP dengan metode ACO
(Bagian 2)**

No	Nama Data	Tipe Data	Keterangan
11	probabilitas	<i>Double</i>	Berisi nilai probabilitas untuk menentukan <i>nodes</i> selanjutnya dalam perjalanan semut
12	Anttour	<i>Integer</i>	Matriks yang berisi daftar perjalanan semut menyelesaikan TSP
13	Cost	<i>Double</i>	Matriks yang menyimpan hasil dari proses penyelesaian TSP menggunakan GA
14	Terpendek	<i>Double</i>	Berisi nilai <i>cost</i> yang paling minimal dari setiap iterasi
15	palingpendek	<i>Double</i>	Berisi nilai <i>terpendek</i> yang minimal untuk keseluruhan iterasi.

3.1.2.3. Data Proses *Pairwise Comparison Matrix*

Data proses untuk metode *Pairwise Comparison Matrix* terdapat pada Tabel 3.1-7 sampai dengan Tabel 3.1-9.

Tabel 3.1-7 Data Proses Penyelesaian Pairwise Comparison Matrix (Bagian 1)

No	Nama Data	Tipe Data	Keterangan
1	A	<i>Double</i>	Matriks yang menyimpan nilai dari kriteria yang ingin dibandingkan, nilai tersebut berdasarkan skala perbandingan yang telah ditentukan
2	Jumlah	<i>Integer</i>	Jumlah kriteria yang dibandingkan
3	RI	<i>Double</i>	Menyimpan hasil perhitungan <i>acak index</i>
4	normalizedA	<i>Double</i>	Matriks yang menyimpan hasil normalisasi dari matrix A
5	U	<i>Double</i>	Nilai yang menyimpan bobot vektor dari matriks A
6	N	<i>Double</i>	Matriks yang menyimpan nilai bobot dari matriks normalisasi matriks A
7	nmax	<i>Double</i>	Nilai yang menyimpan total penjumlahan bobot dari matriks normalisasi matriks A digunakan untuk mencari nilai <i>CI</i>
8	CI	<i>Double</i>	Menyimpan hasil perhitungan nilai <i>confidence index</i>

Tabel 3.1-8 Data Proses Penyelesaian Pairwise Comparison Matrix (Bagian 2)

No	Nama Data	Tipe Data	Keterangan
9	CR	Double	Menyimpan hasil perhitungan nilai <i>confidence ratio</i> untuk melihat kelayakan dari matriks A
10	Awaktu	Double	Matriks turunan untuk membandingkan kriteria waktu dari algoritma yang dibandingkan
11	Amemori	Double	Matriks turunan untuk membandingkan kriteria memori dari algoritma yang dibandingkan
12	Ajarak	Double	Matriks turunan untuk membandingkan kriteria jarak dari algoritma yang dibandingkan
13	Akode	Double	Matriks turunan untuk membandingkan kriteria jumlah baris kode dari algoritma yang dibandingkan
14	jumlahturunan	integer	Jumlah kriteria yang dibandingkan
15	normalizedA1, normalizedA2, normalizedA3, normalizedA4	Double	Matriks yang menyimpan hasil normalisasi dari matrix turunan

Tabel 3.1-9 Data Proses Penyelesaian Pairwise Comparison Matrix (Bagian 3)

No	Nama Data	Tipe Data	Keterangan
16	U1, U2, U3, U4	Double	Nilai yang menyimpan bobot vektor dari matriks turunan
17	PGA	Double	Nilai yang menyimpan hasil akhir dari nilai yang didapatkan oleh algoritma GA
18	PACO	Double	Nilai yang menyimpan hasil akhir dari nilai yang didapatkan oleh algoritma ACO

3.1.3. Data Keluaran

Data keluaran yang dihasilkan dari proses penyelesaian TSP menggunakan kedua metode kecerdasan buatan tersebut dapat dibagi menjadi tiga bagian yaitu jarak tempuh terpendek, ruang memori yang dipakai, total *running time program*. Jarak tempuh terpendek adalah hasil penjumlahan jarak yang ditempuh *salesman*. Ruang memori yang dipakai merupakan hasil dari penyimpanan variabel proses-proses yang dilalui. Sedangkan total *running time program* adalah waktu yang dibutuhkan untuk menyelesaikan sebuah permasalahan TSP.

3.2. Proses Penyelesaian TSP

Bagian ini akan menjelaskan proses pemilihan metode dalam GA dan ACO untuk penyelesaian TSP. Penjelasan akan dibagi menjadi 4 bagian yaitu desain metode, notasi yang digunakan, penentuan total jarak dan batasan algoritma untuk masing-masing algoritma.

3.2.1. Proses Penyelesaian TSP dengan GA

Bagian ini akan menjelaskan proses pemilihan metode dalam GA untuk penyelesaian TSP.

3.2.1.1. Desain Metode GA

Dalam proses penyelesaian TSP menggunakan GA ini ada beberapa tahap pemrosesan yaitu inisialisasi data awal, proses pembangkitan populasi awal dan proses membangkitkan generasi baru.

Inisialisasi data awal adalah proses pengolahan data masukan yaitu data kordinat *nodes*, data jumlah populasi, data jumlah iterasi, probabilitas kawin silang, probabilitas mutasi. Proses pembangkitan awal adalah proses mendapatkan sejumlah kromosom yang selanjutnya dinamakan populasi, populasi ini kemudian akan diproses hingga memperoleh hasil yang diinginkan. Proses pembangkitan generasi baru merupakan kelanjutan dari proses pembangkitan populasi awal, di sini objek yang diproses adalah populasi, sehingga dari poplasi tersebut dihasilkan sejumlah kromosom yang selanjutnya dinamakan generasi, tentunya dalam membangkitkan generasi ini diperlukan beberapa proses seperti mutasi, seleksi, dan *crossover*. Desain metode GA secara umum dapat dilihat pada diagram alir dalam Gambar 3.2-1.

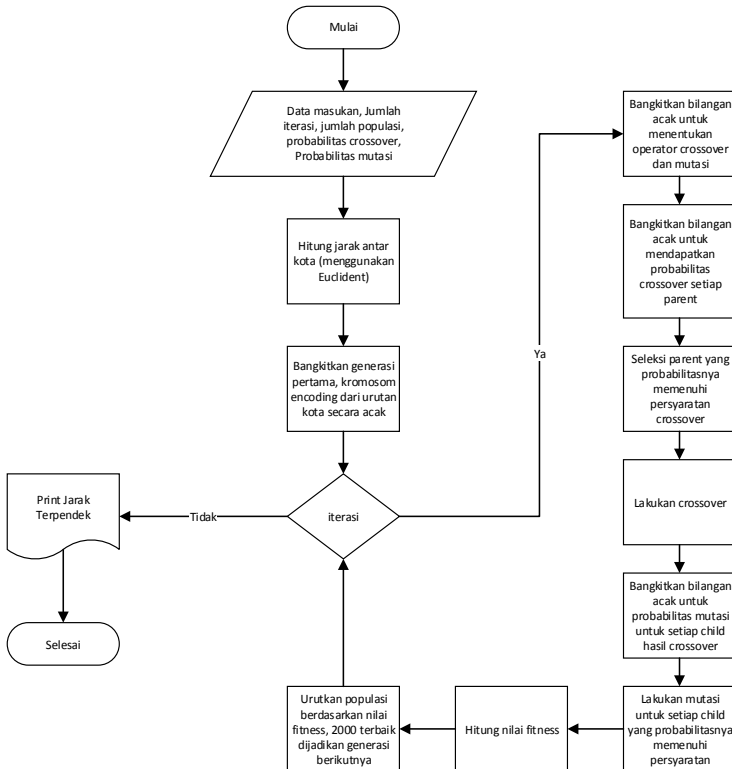
3.2.1.2. Penentuan Total Jarak Terpendek

Bagian ini akan menjelaskan proses penyelesaian TSP menggunakan GA. Berikut urutan-urutan proses yang harus dilalui:

1. Bangkitkan Generasi Pertama

Pada proses ini kita bangkitkan sejumlah individu yang akan membentuk sebuah populasi. Setiap individu atau

kromosom akan mempunyai gen yang berjumlah sama dengan jumlah *nodes*.



Gambar 3.2-1 Diagram Alir Proses Penyelesaian TSP dengan Metode GA

Gen pada setiap kromosom mengalami pengodean sesuai dengan nomor *nodes* pada data masukan yang diurutkan secara acak.

2. Bangkitkan Operator Kawin Silang dan Operator Mutasi

Pada proses ini kita bangkitkan dua operator kawin silang secara acak, dan dua operator mutasi secara acak. Untuk membangkitkan dua operator kawin silang ditetapkan beberapa ketentuan yaitu bilangan harus positif integer dan $1 \leq c_1 < c_2 \leq \text{jumlah}$. Sementara untuk membangkitkan dua operator mutasi ketentuan yang harus dipatuhi yaitu bilangan harus positif dan $1 \leq m_1 \mid m_2 \leq \text{jumlah}$.

3. Seleksi *Parent* Untuk Proses Kawin Silang

Pada proses ini kita akan menyeleksi *parent* sebelum memasuki proses *crossover*. Sebelumnya kita akan membangkitkan bilangan acak untuk setiap individu yang akan menjadi calon *parent*. Hasil pembangkitan bilangan acak tersebut akan dibandingkan dengan probabilitas *crossover* yang telah ditentukan sebelumnya dengan nilai 0.5. Setiap calon *parent* yang memiliki nilai hasil pembangkitan secara acak lebih rendah dari probabilitas *crossover* yang telah ditentukan $\text{parentPCO} < \text{pcO}$, terpilih menjadi *parent* yang akan mengalami proses *crossover*. Simpan indeks setiap *parent* yang terpilih ke dalam sebuah list.

Setiap proses kawin silang dibutuhkan dua *parent*. Proses pemilihan dua *parent* ini ditentukan berdasarkan urutan pada list yang menyimpan indeks setiap *parent* yang terpilih. Dengan demikian dapat dikatakan *parent* pertama pada list akan dikawin silangkan dengan *parent* kedua pada list, *parent* kedua pada list akan dikawin silangkan dengan *parent* ketiga pada list, begitu seterusnya hingga *parent* terakhir pada list akan dikawin silangkan dengan *parent* sebelumnya.

4. Proses Kawin Silang

Pada tahap ini akan dijelaskan proses kawin silang terjadi. Melanjuti proses sebelumnya, setelah didapatkan dua *parent* maka akan dilakukan proses kawin silang ini. Proses kawin silang ini sendiri memiliki tujuan yaitu untuk mendapat keturunan atau yang biasa kita sebut *child*.

Langkah pertama pada proses kawin silang ini adalah menukarkan posisi $c_1, c_1 + 1, \dots, c_2$ milik $parent_i$ dengan milik $parent_{i+1}$. Simpan proses langkah pertama pada sebuah list dan beri nama list tersebut dengan nama *child*. Langkah berikutnya adalah membuat list baru L_i, L_{i+1} dan mengisi list tersebut dengan elemen-elemen yang dimiliki oleh $parent_i$ dan $parent_{i+1}$ dimulai dari c_2 searah jarum jam $c_2 + 1, c_2 + 2, \dots, n, 1, 2, \dots, c_2$. Langkah ketiga merupakan lanjutan langkah sebelumnya, setelah kita memiliki list L, kita buat sebuah list baru lagi yang diberi nama dengan L'_i, L'_{i+1} , dan mengisi list tersebut dengan menghapus gen yang nilai nodes-nya telah termasuk dalam *child* sementara. Dan proses terakhir adalah melanjutkan pengisian pada variabel *child*, dengan cara mengambil elemen-elemen pada variabel L'_i, L'_{i+1} dan mengurutkannya secara $c_2 + 1, c_2 + 2, \dots, n, 1, 2, \dots, c_1 - 1$. Untuk lebih jelasnya proses kawin silang akan dijelaskan pada Tabel 3.2-1 [7].

5. Seleksi Child Untuk Proses Mutasi

Pada proses ini kita akan menyeleksi *child* sebelum memasuki proses mutasi. Sebelumnya kita akan membangkitkan bilangan acak untuk setiap *child*. Hasil pembangkitan bilangan acak tersebut akan dibandingkan dengan probabilitas mutasi yang telah ditentukan sebelumnya dengan nilai 0.01. Setiap

child yang memiliki nilai hasil pembangkitan secara acak lebih rendah dari probabilitas mutasi yang telah ditentukan $childPM < pm$, terpilih menjadi *child* yang akan mengalami proses mutasi. Simpan indeks setiap *child* yang terpilih ke dalam sebuah list.

Tabel 3.2-1 Urutan Proses Crossover yang Terpilih

Urutan	Proses	Contoh
1	Tentukan $parent_1$ dan $parent_2$ dari populasi	$P1 = 1-2-5-4-3-7-6$ $P2 = 5-4-2-6-3-1-7$
2	Tentukan titik <i>crossover</i> secara acak	$c_1 = 3$ $c_2 = 5$
3	Tukar posisi $c_1, c_1 + 1, \dots, c_2$ milik $parent_1$ dengan milik $parent_2$ dan simpan kedalam variabel <i>child1</i> dan <i>child2</i>	$child1 = ?-?-2-6-3-?-?$ $child2 = ?-?-5-4-3-?-?$
4	Buat list baru L_1, L_2 dan mengisinya dengan element $parent_1$ dan $parent_2$ dimulai dari c_2 searah jarum jam $c_2 + 1, c_2 + 2, \dots, n, 1, 2, \dots, c_2$	$L1 = 7,6,1,2,5,4,3$ $L2 = 1,7,5,4,2,6,3$
5	Dari list L_1, L_2 buat sebuah list baru lagi L'_1, L'_2 dengan menghapus <i>nodes</i> yang telah termasuk dalam c_1 pada tahap 2	$L'_1 = L1 - 2,6,3 = 7,1,5,4$ $L'_2 = L2 - 5,4,3 = 1,7,2,6$
6	Terakhir masukan setiap element L'_i kedalam element kosong c_i dengan urutan $c_2 + 1, c_2 + 2, \dots, n, 1, 2, \dots, c_{1-1}$	$child1 = 5-4-2-6-3-7-1$ $child2 = 2-6-5-4-3-1-7$

6. Proses Mutasi

Melalui proses sebelumnya kita mempunyai list yang menyimpan indeks setiap *child* yang akan mengalami

mutasi, yang berarti setiap *child* tersebut akan mengalami mutasi. Proses mutasi yang akan dilakukan pada tahap ini adalah proses menukar posisi kromosom yang telah ditentukan berdasarkan proses inisialisasi data. Yaitu kromosom yang berada pada posisi m_1 bertukar tempat dengan kromosom yang berada pada posisi m_2 .

7. Proses Menghitung *Fitness* Setiap Individu

Pada tahap ini akan dilakukan proses menghitung nilai *fitness* setiap individu. Cara menghitung nilai *fitness* setiap individu sudah ditentukan dengan nilai *fitness* sama dengan jarak tempuh setiap individu. Berarti nilai *fitness* didapatkan dengan melihat kromosom setiap individu.

8. Proses Elitisme

Pada tahap ini akan dilakukan proses elitisme yang berarti proses perbaikan generasi. Proses ini dilakukan dengan mengevaluasi setiap individu berdasarkan nilai *fitness*-nya. Setiap individu *parent* maupun *child* diurutkan berdasarkan nilai *fitness*. Sejumlah populasi, setiap individu yang memiliki nilai *fitness* terbaik akan menjadi generasi baru pada iterasi berikutnya.

3.2.2. Proses Penyelesaian TSP dengan ACO

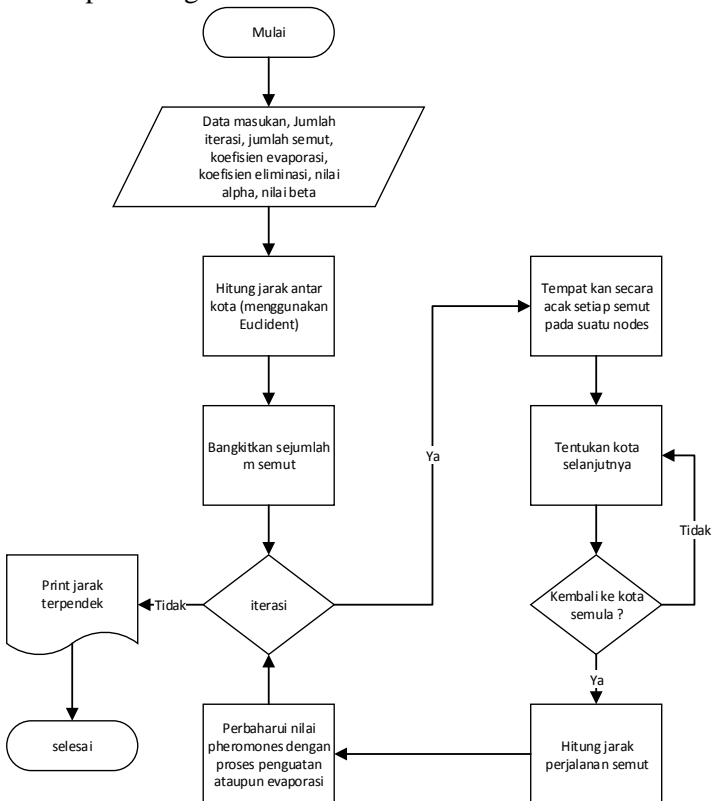
Bagian ini akan menjelaskan proses pemilihan metode dalam GA untuk penyelesaian TSP.

3.2.2.1. Desain Metode ACO

Dalam proses penyelesaian TSP menggunakan ACO ini ada beberapa tahap pemrosesan, yaitu inisialisasi data awal, penentuan jalur yang dilewati semut, memperbarui nilai *pheromones*, menghitung semua jalur yang dilewati setiap semut.

Inisialisasi data awal adalah proses pengolahan data masukan yaitu data koordinat *nodes*, data jumlah semut, data jumlah iterasi, data koefisien *alpha*, *beta*, eliminasi, dan evaporasi. Penentuan jalur yang dilewati semut adalah proses

menentukan perjalanan setiap *nodes* yang akan dilewati semut berdasarkan nilai *pheromones* sementara. Memperbarui nilai *pheromones* adalah proses dimana *pheromones* pada jalur yang dilalui semut mengalami penguatan, sebaliknya *pheromones* mengalami evaporasi jika jalur tersebut tidak dilalui. Terakhir adalah proses menghitung *cost* pada jalur yang dilalui semut. Desain metode ACO secara umum dapat dilihat pada diagram alir dalam Gambar 3.2-2.



Gambar 3.2-2 Diagram Alir Proses Penyelesaian TSP dengan Metode ACO

3.2.2.2. Penentuan Total Jarak Terpendek

Bagian ini akan menjelaskan proses penyelesaian TSP menggunakan GA. Berikut urutan-urutan proses yang harus dilalui:

1. Bangkitkan Sejumlah Semut

Pada tahap ini kita membangkitkan sejumlah semut sesuai dengan yang kita inginkan. Disini peran semut dianggap seperti seorang *salesman*. Semut akan berjalan dari tempat awal dan kembali ke tempat semula setelah melalui semua kota.

2. Tentukan *Entry State* untuk Setiap Semut

Tahap ini merupakan lanjutan dari tahap sebelumnya. Pada tahap ini kita harus menempatkan setiap semut pada sebuah kota yang ditentukan secara acak. Guna penempatan ini adalah kota yang ditentukan menjadi *entry state* setiap semut, dimana seekor semut akan memulai dan mengakhir perjalanan pada kota tersebut.

3. Tentukan *Next State* atau Perjalanan Semut

Pada tahap ini seekor semut akan menentukan *nodes* berikutnya untuk mencapai tempat tujuan. Setiap semut akan memiliki probabilitas yang sudah dibahas pada bab sebelumnya untuk menentukan *nodes* berikutnya. Dengan membangkitkan bilangan acak untuk dibandingkan dengan probabilitas yang dimiliki semut. Keadaan *pheromones* sementara mempengaruhi nilai probabilitas yang dimiliki semut, yang berarti mempengaruhi pemilihan *nodes* berikutnya. Dapat disimpulkan jalur yang sering dilewati atau yang memiliki nilai *pheromones* besar memiliki kemungkinan lebih besar untuk terpilih.

4. Evaluasi dan Perbaharui Nilai *Pheromones*

Berdasarkan pada tahap sebelumnya setiap semut akan mempunyai jalur yang dilalui masing masing. Tahap ini akan memperbaharui nilai *pheromones* pada semua jalur. Setiap jalur yang dilewati semut, maka *pheromones* tersebut akan mengalami peningkatan. Sebaliknya nilai

pheromones akan menguap jika tidak dilewati semut. Fungsi untuk meningkatkan atau mengurangi nilai *pheromones* sudah dibahas pada bab sebelumnya.

5. Hitung Cost Perjalanan Semut

Tahap ini merupakan tahap turunan dari tahap tiga. Setelah setiap semut mempunyai jalur masing masing untuk menyelesaikan perjalanan. Masing-masing semut memiliki panjang jarak tempuh yang berbeda sesuai jalur yang dilaluinya.

3.2.3. Batasan Algoritma

Pada penerapan algoritma kecerdasan buatan untuk menyelesaikan TSP ini terdapat beberapa batasan. Batasan-batasannya adalah sebagai berikut:

1. Menggunakan metode Euclidean dalam menghitung jarak antar *nodes*

Pada algoritma kecerdasan buatan ini yang menjadi salah satu hasil keluaran adalah jarak tempuh. Berbicara tentang jarak, hal yang penting adalah cara kita mendapatkannya karena ada beberapa metode untuk menghitung jarak. Dan yang dipakai dalam algoritma ini adalah metode Euclidean. Bagaimana cara menghitung jarak menggunakan metode Euclidean dapat dilihat pada Persamaan 3.1.

$$D = \sqrt{\sum_{k=1}^n (px_k - qx_k)^2 + (py_k - qy_k)^2} \quad (3.1)$$

Keterangan:

D = jarak.

n = jumlah *nodes*.

k = atribut kota.

px = absis dari kota p .

qx = absis dari kota q .

py = ordinat dari kota p .

qy = ordinat dari kota q .

Dalam proses perhitungan ini, jarak yang dihitung adalah *path* antar *nodes*.

2. **Menggunakan acak uniform dalam semua proses pengacakan**

Metode *acak* dalam ilmu komputasi itu ada banyak, dan salah satunya adalah *acak uniform*. Dalam pengimplementasian algoritma GA dan ACO ini *acak uniform* sengaja dipilih karena proses pengacakan yang kemudian hasilnya dibandingkan dengan probabilitas mutasi dan *crossover* pada GA dan dibandingkan dengan probabilitas *next state* pada ACO harus mempunyai peluang yang sama besar dan seragam di dalam rentang yang sudah ditentukan.

3. **Probabilitas pada operator GA sudah ditentukan**

Ada beberapa probabilitas yang nilainya dibuat konstan pada algoritma ini, di antaranya adalah:

1. Probabilitas *crossover* sebesar 0.5.
2. Probabilitas mutasi sebesar 0.01.

4. **Data masukan pada GA sudah ditentukan**

Batasan selanjutnya adalah mengenai data masukan. Sangat memungkinkan terjadi ketidak sesuaian dengan solusi yang diinginkan jika program ini diberi data masukan yang berbeda, sehingga untuk menghindari hal itu data masukannya akan ditentukan, yaitu:

1. Jumlah iterasi sebanyak 1000.
2. Jumlah populasi yang dibangkitkan sebanyak 2000.

5. **Data masukan pada ACO sudah ditentukan**

Untuk ACO data masukannya sudah ditentukan, yaitu:

1. Jumlah iterasi sebanyak 100.
2. Jumlah semut yang dibangkitkan sebanyak 200.

3. Nilai alpha adalah 1.
4. Nilai beta adalah 5.
5. Nilai koefisien eliminasi adalah 0.96.
6. Nilai koefisien evaporasi adalah 0.1.

3.3. Proses Pairwise Comparison Matrix

Bagian ini akan menjelaskan proses *Pairwise Comparison Matrix* untuk membandingkan kinerja *variants* dari dua metode kecerdasan buatan yang telah dijelaskan pada subbab sebelumnya dalam menyelesaikan permasalahan TSP.

Untuk dapat menggunakan *Pairwise Comparison Matrix* sebelumnya kita harus membangun implementasi metode GA dan ACO dalam penyelesaian masalah TSP, sehingga kita mempunyai setiap nilai kinerja dari *variants* yang akan dibandingkan. Terdapat empat *variants* atau dapat disebut dengan kriteria yang akan dibandingkan pada proses ini yaitu kompleksitas waktu implementasi, kompleksitas memori implementasi, total jarak tempuh yang paling optimal, dan tingkat kesulitan dalam implementasi yang akan direpresentasikan dengan jumlah baris pada kode implementasi kedua algoritma tersebut.

Sebelum membangun sebuah matriks kita harus memahami terlebih dahulu konsep derajat kepentingan. Derajat kepentingan adalah nilai yang kita berikan untuk kriteria yang akan kita bandingkan. Fungsi dari derajat kepentingan ini adalah mengurutkan tingkat prioritas dari kriteria yang dibandingkan tersebut. Skala derajat kepentingan ditunjukkan pada Tabel 3.3-1.

Tabel 3.2-2 Tabel Penentuan Derajat Kepentingan

Nilai	Tingkat Kepentingan
1	Memiliki tingkatan yang sama
3	Sedikit lebih penting dari salah satu kriteria lainnya
5	Lebih penting
7	Memiliki kepentingan yang sangat tinggi
9	Memiliki tingkat kepentingan yang paling tinggi

Pengisian untuk matriks yang akan dibangun berdasarkan dari Tabel 3.3-1. Nilai 1 diberikan jika kedua kriteria yang dibandingkan memiliki nilai yang sama. Nilai 3 diberikan jika salah satu kriteria lebih unggul dari kriteria lainnya tanpa selisih yang besar atau kurang dari dua kali lipat. Nilai 5 diberikan hanya jika salah satu kriteria memiliki keunggulan lebih dari dua kali lipat tapi tidak melebihi tiga kali lipat dari kriteria lain. Sedangkan nilai 7 diberikan hanya jika salah satu kriteria memiliki keunggulan lebih dari tiga kali lipat tapi tidak melebihi empat kali lipat dari kriteria lain. Dan nilai 9 diberikan hanya jika salah satu kriteria unggul dari kriteria lain dengan selisih melebihi dari empat kali lipat. Sebelum kita membandingkan kedua algoritma kecerdasan buatan yang telah disebutkan sebelumnya. Kita harus membangun sebuah matriks utama *A* yang akan berfungsi sebagai matriks yang akan membandingkan nilai derajat kepentingan dari keempat *varians* yang akan dibandingkan. Penomoran untuk keempat *varians* tersebut akan ditunjukkan pada Tabel 3.3-2. Yang menjadi kriteria satu adalah kompleksitas waktu implementasi, yang menjadi kriteria kedua adalah kompleksitas memori implementasi, yang menjadi kriteria ketiga adalah jarak tempuh optimal, dan yang menjadi kriteria terakhir adalah tingkat kesulitan dalam implementasi. Sedangkan penomoroan untuk kedua algoritma

kecerdasan buatan yang diimplementasikan ditunjukkan pada Tabel 3.3-3. GA akan menjadi kriteria nomor satu, sedangkan ACO yang akan menjadi nomor dua.

Tabel 3.2-3 Urutan Kriteria Pembandingan

1	Kompleksitas Waktu Implementasi
2	Kompleksitas Memori Implementasi
3	Jarak Tempuh Optimal
4	Tingkat Kesulitan Implementasi

Tabel 3.2-4 Urutan Algoritma Pembandingan

1	<i>Basic Genetic Algorithm</i>
2	<i>Basic Ant Colony Optimization</i>

Matriks utama A akan membandingkan derajat kepentingan dari keempat *varians* yang telah disebutkan. Tidak membutuhkan hasil keluaran dari implementasi penyelesaian TSP menggunakan kedua metode kecerdasan buatan. Matriks utama A ini hanya digunakan untuk mendapatkan nilai *consistency index* yang akan menjadi nilai keseimbangan dari proses *pairwise comparison* ini. Proses pembentukan matriks A akan ditunjukkan pada Persamaan 3.2.

$$A = \begin{pmatrix} 1 & c_1/c_2 & c_1/c_3 & c_1/c_4 \\ c_2/c_1 & 1 & c_2/c_3 & c_2/c_4 \\ c_3/c_1 & c_3/c_2 & 1 & c_3/c_4 \\ c_4/c_1 & c_4/c_2 & c_1/c_3 & 1 \end{pmatrix} \quad (3.2)$$

Urutan tingkat kepentingannya adalah jarak optimal, kompleksitas waktu, kompleksitas memori, dan jumlah kode implementasi. Sehingga didapatkan matriks A adalah sebagai berikut.

$$A = \begin{pmatrix} 1.00 & 5.00 & 0.33 & 7.00 \\ 0.20 & 1.00 & 0.14 & 3.00 \\ 3.00 & 7.00 & 1.00 & 9.00 \\ 0.14 & 0.33 & 0.11 & 1.00 \end{pmatrix}$$

Selanjutnya setelah mendapatkan matriks A kita buat sebuah matriks baru yang merupakan hasil normalisasi dari matriks A . Normalisasi matrix A didapatkan dengan pembagian setiap elemen-elemen dari matrix A dengan total penjumlahan dari kolom elemen tersebut. Contoh untuk mendapatkan kolom pertama baris pertama dari normalisasi matrix A maka dilakukan perhitungan 1 dibagi dengan $(1 + 0.2 + 3 + 0.14) = 0.23$, begitu seterusnya untuk elemen-elemen lainnya. Dari proses ini didapatkan normalisasi matrix A sebagai berikut.

$$N_A = \begin{pmatrix} 0.23 & 0.375 & 0.21 & 0.35 \\ 0.04 & 0.075 & 0.09 & 0.15 \\ 0.70 & 0.525 & 0.63 & 0.45 \\ 0.03 & 0.025 & 0.07 & 0.05 \end{pmatrix}$$

Dari matriks baru tersebut kita akan mendapatkan nilai bobot vektor dan dapat dicari nilai CI dan CR yang fungsinya sudah dijelaskan pada bab sebelumnya. Bobot vektor didapatkan mencari nilai rata-rata dari setiap baris dari normalisasi matrix A . Proses untuk mendapatkan bobot vektor ditunjukkan sebagai berikut.

$$\begin{aligned} U_{A1} &= \frac{0.23 + 0.375 + 0.21 + 0.35}{4} = 0.29 \\ U_{A2} &= \frac{0.04 + 0.075 + 0.09 + 0.15}{4} = 0.09 \\ U_{A3} &= \frac{0.70 + 0.525 + 0.63 + 0.45}{4} = 0.57 \\ U_{A4} &= \frac{0.03 + 0.025 + 0.07 + 0.05}{4} = 0.04 \end{aligned}$$

Untuk koefisien nilai RI ditunjukkan pada Persamaan 3.3.

$$RI = \frac{1.96(n - 2)}{n} \quad (3.3)$$

Sehingga didapatkan nilai $RI = 0.99$. Langkah selanjutnya adalah mencari nilai CI . Untuk mendapat nilai CI kita membutuhkan nilai n_{max} . Nilai n_{max} didapatkan dari proses pengalaian matriks A dengan matriks bobot vektor, sehingga didapatkan sebuah matrix baru berukuran 1×4 . Hasil penjumlahan dari matrix baru tersebut merupakan nilai n_{max} . Sehingga $n_{max} = 4.27$. Dengan persamaan yang telah disebutkan pada bab sebelumnya kita mendapatkan nilai $CI = 0.09$. Proses pembagian nilai CI dengan nilai RI mendapatkan nilai $CR = 0.09$, sehingga *matrix A dapat diterima keseimbangannya*.

Setelah mendapat nilai bobot vektor dari matriks A , kita harus membuat matriks baru (matriks turunan dari matriks A) lagi sejumlah kriteria yang ingin dibandingkan dalam kasus ini terdapat 4 matriks baru. Dari setiap matriks yang baru dibuat, kita cari kembali nilai bobot vektor masing masing matriks.

Untuk dapat membuat matriks turunan kita memerlukan pengetahuan akan nilai dari setiap kriteria yang kita ingin bandingkan. Nilai dari setiap kriteria, yaitu kompleksitas waktu, kompleksitas memori, dan jarak optimal merujuk pada hasil uji coba Bab 5, sedangkan tingkat kesulitan implementasi merujuk pada total baris kode pada Bab 4.

Terakhir setelah kita mendapatkan nilai bobot vektor dari matriks A dan nilai bobot vektor dari keempat matriks kriteria maka kita dapat menghitung total nilai (*points*) yang didapatkan oleh masing-masing algoritma yang dibandingkan.

Berikut adalah hasil proses perhitungan manual dari matriks turunan, bobot vektor, dan total nilai dari kedua algoritma kecerdasan buatan yang dibandingkan berdasarkan 5 uji coba yang dilakukan.

3.3.1. Uji Coba 14 Nodes

Berdasarkan hasil uji coba pada Bab 5, nilai dari setiap kriteria (masing-masing *varians*) dari kedua algoritma pada uji coba 14 *nodes* ditunjukkan pada Tabel 3.3-4.

Tabel 3.2-5 Tabel Hasil Uji Coba 14 Nodes

Percobaan	14 Nodes	
Metode	GA	ACO
Kompleksitas waktu	414.27s	3.22s
Kompleksitas memori	45.3 MB	1.35 MB
Jarak optimal	3087.5	3087.5
Baris kode	261 baris	152 baris

Dari data diatas kita dapat membuat 4 matriks turunan baru, yang dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 A_{waktu} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
 A_{memori} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
 A_{jarak} &= \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \\
 A_{kode} &= \begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}
 \end{aligned}$$

Dengan proses yang sama seperti pada proses matrix A, langkah selanjutnya adalah membuat matriks normalisasi dari 4 matriks turunan tersebut. Sehingga normalisasi dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
N_{A_{waktu}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
N_{A_{memori}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
N_{A_{jarak}} &= \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix} \\
N_{A_{kode}} &= \begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}
\end{aligned}$$

Dengan proses yang sama bobot vektor dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
UA_{waktu} &= (0.1; 0.9)^Z \\
UA_{memori} &= (0.1; 0.9)^Z \\
UA_{jarak} &= (0.5; 0.5)^Z \\
UA_{kode} &= (0.25; 0.75)^Z
\end{aligned}$$

Dari proses proses yang telah dilakukan kita dapat menghitung nilai akhir dari GA dan ACO. Nilai P dari dua algoritma tersebut dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
P(GA) &= (0.29 \times 0.1) + (0.09 \times 0.1) + (0.57 \times 0.5) \\
&\quad + (0.04 \times 0.25) = 0.334 \\
P(ACO) &= (0.29 \times 0.9) + (0.09 \times 0.9) + (0.57 \times 0.5) \\
&\quad + (0.04 \times 0.75) = 0.666
\end{aligned}$$

3.3.2. Uji Coba 16 Nodes

Berdasarkan hasil uji coba pada Bab 5, nilai dari setiap kriteria (masing-masing *varians*) dari kedua algoritma pada uji coba 16 *nodes* ditunjukkan pada Tabel 3.3-5.

Tabel 3.2-6 Tabel Hasil Uji Coba 16 Nodes

Percobaan	16 Nodes	
Metode	GA	ACO
Kompleksitas waktu	385.06s	4.09s
Kompleksitas memori	45.9 MB	1.94 MB
Jarak optimal	7398	7398
Baris kode	261 baris	152 baris

Dari data diatas kita dapat membuat 4 matriks turunan baru, yang dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 A_{waktu} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
 A_{memori} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
 A_{jarak} &= \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \\
 A_{kode} &= \begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}
 \end{aligned}$$

Dengan proses yang sama seperti pada proses matrix A , langkah selanjutnya adalah membuat matriks normalisasi dari 4 matriks turunan tersebut. Sehingga normalisasi dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 N_{A_{waktu}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
 N_{A_{memori}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
 N_{A_{jarak}} &= \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix} \\
 N_{A_{kode}} &= \begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}
 \end{aligned}$$

Dengan proses yang sama bobot vektor dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
UA_{waktu} &= (0.1; 0.9)^Z \\
UA_{memori} &= (0.1; 0.9)^Z \\
UA_{jarak} &= (0.5; 0.5)^Z \\
UA_{kode} &= (0.25; 0.75)^Z
\end{aligned}$$

Dari proses proses yang telah dilakukan kita dapat menghitung nilai akhir dari GA dan ACO. Nilai P dari dua algoritma tersebut dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
P(GA) &= (0.29 \times 0.1) + (0.09 \times 0.1) + (0.57 \times 0.5) \\
&\quad + (0.04 \times 0.25) = 0.334 \\
P(ACO) &= (0.29 \times 0.9) + (0.09 \times 0.9) + (0.57 \times 0.5) \\
&\quad + (0.04 \times 0.75) = 0.666
\end{aligned}$$

3.3.3. Uji Coba 22 Nodes

Berdasarkan hasil uji coba pada Bab 5, nilai dari setiap kriteria (masing-masing *varians*) dari kedua algoritma pada uji coba 22 *nodes* ditunjukkan pada Tabel 3.3-6.

Tabel 3.2-7 Tabel Hasil Uji Coba 22 Nodes

Percobaan	22 Nodes	
Metode	GA	ACO
Kompleksitas waktu	500.91s	6.1s
Kompleksitas memori	50.9 MB	2.61 MB
Jarak optimal	7530.97	7601
Baris kode	261 baris	152 baris

Dari data diatas kita dapat membuat 4 matriks turunan baru, yang dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
A_{waktu} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
A_{memori} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
A_{jarak} &= \begin{pmatrix} 1 & 3 \\ 0.33 & 1 \end{pmatrix} \\
A_{kode} &= \begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}
\end{aligned}$$

Dengan proses yang sama seperti pada proses matrix A , langkah selanjutnya adalah membuat matriks normalisasi dari 4 matriks turunan tersebut. Sehingga normalisasi dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
N_{A_{waktu}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
N_{A_{memori}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
N_{A_{jarak}} &= \begin{pmatrix} 0.75 & 0.75 \\ 0.25 & 0.25 \end{pmatrix} \\
N_{A_{kode}} &= \begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}
\end{aligned}$$

Dengan proses yang sama bobot vektor dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
UA_{waktu} &= (0.1; 0.9)^Z \\
UA_{memori} &= (0.1; 0.9)^Z \\
UA_{jarak} &= (0.75; 0.25)^Z \\
UA_{kode} &= (0.25; 0.75)^Z
\end{aligned}$$

Dari proses proses yang telah dilakukan kita dapat menghitung nilai akhir dari GA dan ACO. Nilai P dari dua algoritma tersebut dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 P(GA) &= (0.29 \times 0.1) + (0.09 \times 0.1) \\
 &\quad + (0.57 \times 0.75) \\
 &\quad + (0.04 \times 0.25) = 0.4765 \\
 P(ACO) &= (0.29 \times 0.9) + (0.09 \times 0.9) \\
 &\quad + (0.57 \times 0.25) \\
 &\quad + (0.04 \times 0.75) = 0.5235
 \end{aligned}$$

3.3.4. Uji Coba 51 Nodes

Berdasarkan hasil uji coba pada Bab 5, nilai dari setiap kriteria (masing-masing *varians*) dari kedua algoritma pada uji coba 51 nodes ditunjukkan pada Tabel 3.3-7.

Tabel 3.2-8 Tabel Hasil Uji Coba 51 Nodes

Percobaan	51 Nodes	
Metode	GA	ACO
Kompleksitas waktu	1536.62s	19.35s
Kompleksitas memori	82.8 MB	10.1 MB
Jarak optimal	466.4	501.98
Baris kode	261 baris	152 baris

Dari data diatas kita dapat membuat 4 matriks turunan baru, yang dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 A_{waktu} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
 A_{memori} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
 A_{jarak} &= \begin{pmatrix} 1 & 3 \\ 0.33 & 1 \end{pmatrix} \\
 A_{kode} &= \begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}
 \end{aligned}$$

Dengan proses yang sama seperti pada proses matrix A, langkah selanjutnya adalah membuat matriks normalisasi

dari 4 matriks turunan tersebut. Sehingga normalisasi dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 N_{Awaktu} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
 N_{A_{memori}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
 N_{A_{jarak}} &= \begin{pmatrix} 0.75 & 0.75 \\ 0.25 & 0.25 \end{pmatrix} \\
 N_{A_{kode}} &= \begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}
 \end{aligned}$$

Dengan proses yang sama bobot vektor dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 UA_{waktu} &= (0.1; 0.9)^Z \\
 UA_{memori} &= (0.1; 0.9)^Z \\
 UA_{jarak} &= (0.75; 0.25)^Z \\
 UA_{kode} &= (0.25; 0.75)^Z
 \end{aligned}$$

Dari proses proses yang telah dilakukan kita dapat menghitung nilai akhir dari GA dan ACO. Nilai P dari dua algoritma tersebut dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 P(GA) &= (0.29 \times 0.1) + (0.09 \times 0.1) \\
 &\quad + (0.57 \times 0.75) \\
 &\quad + (0.04 \times 0.25) = 0.4765 \\
 P(ACO) &= (0.29 \times 0.9) + (0.09 \times 0.9) \\
 &\quad + (0.57 \times 0.25) \\
 &\quad + (0.04 \times 0.75) = 0.5235
 \end{aligned}$$

3.3.1. Uji Coba 52 Nodes

Berdasarkan hasil uji coba pada Bab 5, nilai dari setiap kriteria (masing-masing *varians*) dari kedua algoritma pada uji coba 52 *nodes* ditunjukkan pada Tabel 3.3-8.

Tabel 3.2-9 Tabel Hasil Uji Coba 52 Nodes

Percobaan	52 Nodes	
Metode	GA	ACO
Kompleksitas waktu	1500.31s	19.9s
Kompleksitas memori	98.3 MB	11.3 MB
Jarak optimal	8607	9275
Baris kode	261 baris	152 baris

Dari data diatas kita dapat membuat 4 matriks turunan baru, yang dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 A_{waktu} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
 A_{memori} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
 A_{jarak} &= \begin{pmatrix} 1 & 3 \\ 0.33 & 1 \end{pmatrix} \\
 A_{kode} &= \begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}
 \end{aligned}$$

Dengan proses yang sama seperti pada proses matrix A, langkah selanjutnya adalah membuat matriks normalisasi dari 4 matriks turunan tersebut. Sehingga normalisasi dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
 N_{Awaktu} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
 N_{A_{memori}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
 N_{A_{jarak}} &= \begin{pmatrix} 0.75 & 0.75 \\ 0.25 & 0.25 \end{pmatrix} \\
 N_{A_{kode}} &= \begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}
 \end{aligned}$$

Dengan proses yang sama bobot vektor dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
UA_{waktu} &= (0.1; 0.9)^Z \\
UA_{memori} &= (0.1; 0.9)^Z \\
UA_{jarak} &= (0.75; 0.25)^Z \\
UA_{kode} &= (0.25; 0.75)^Z
\end{aligned}$$

Dari proses proses yang telah dilakukan kita dapat menghitung nilai akhir dari GA dan ACO. Nilai P dari dua algoritma tersebut dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
P(GA) &= (0.29 \times 0.1) + (0.09 \times 0.1) \\
&\quad + (0.57 \times 0.75) \\
&\quad + (0.04 \times 0.25) = 0.4765 \\
P(ACO) &= (0.29 \times 0.9) + (0.09 \times 0.9) \\
&\quad + (0.57 \times 0.25) \\
&\quad + (0.04 \times 0.75) = 0.5235
\end{aligned}$$

3.3.2. Uji Coba 76 Nodes

Berdasarkan hasil uji coba pada Bab 5, nilai dari setiap kriteria (masing-masing *varians*) dari kedua algoritma pada uji coba 14 *nodes* ditunjukkan pada Tabel 3.3-8.

Tabel 3.2-10 Tabel Hasil Uji Coba 76 Nodes

Percobaan	52 Nodes	
Metode	GA	ACO
Kompleksitas waktu	1096.47s	38.66s
Kompleksitas memori	105 MB	20.6 MB
Jarak optimal	767.94	736.08
Baris kode	261 baris	152 baris

Dari data diatas kita dapat membuat 4 matriks turunan baru, yang dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
A_{waktu} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
A_{memori} &= \begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix} \\
A_{jarak} &= \begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix} \\
A_{kode} &= \begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}
\end{aligned}$$

Dengan proses yang sama seperti pada proses matrix A , langkah selanjutnya adalah membuat matriks normalisasi dari 4 matriks turunan tersebut. Sehingga normalisasi dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
N_{A_{waktu}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
N_{A_{memori}} &= \begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix} \\
N_{A_{jarak}} &= \begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix} \\
N_{A_{kode}} &= \begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}
\end{aligned}$$

Dengan proses yang sama bobot vektor dari matriks turunan dapat ditunjukkan sebagai berikut.

$$\begin{aligned}
UA_{waktu} &= (0.1; 0.9)^Z \\
UA_{memori} &= (0.1; 0.9)^Z \\
UA_{jarak} &= (0.25; 0.75)^Z \\
UA_{kode} &= (0.25; 0.75)^Z
\end{aligned}$$

Dari proses proses yang telah dilakukan kita dapat menghitung nilai akhir dari GA dan ACO. Nilai P dari dua algoritma tersebut dapat ditunjukkan sebagai berikut.

$$\begin{aligned}P(GA) &= (0.29 \times 0.1) + (0.09 \times 0.1) \\&\quad + (0.57 \times 0.25) \\&\quad + (0.04 \times 0.25) = 0.1915 \\P(ACO) &= (0.29 \times 0.9) + (0.09 \times 0.9) \\&\quad + (0.57 \times 0.75) \\&\quad + (0.04 \times 0.75) = 0.8085\end{aligned}$$

BAB IV IMPLEMENTASI

4.1. Program *Pairwise Comparison Matrix*

Pada program ini akan dibentuk matriks yang berfungsi sebagai alat untuk pembandingan kinerja 4 *varians* dari metode kecerdasan buatan dalam penyelesaian masalah TSP.

4.1.1. Proses Pembentukan Matriks A

Pada tahapan ini akan dilakukan proses pembentukan matriks A yang sudah dijelaskan pada bab sebelumnya. Matriks A berukuran 4 x 4, dimana setiap baris dan kolom merupakan perbandingan satu lawan satu setiap kriteria yang akan dibandingkan. Proses pembentukan matriks A dapat dilihat pada Kode Sumber 4.1-1.

1	A = [1 5 0.3333 7;
2	0.2 1 0.1428 3;
3	3 7 1 9;
4	0.1428 0.3333 0.1111 1];

Kode Sumber 4.1-1 Kode Sumber Pembentukan Matriks A

4.1.2. Perhitungan RI

Pada tahapan ini akan dilakukan proses perhitungan nilai RI yang sudah dijelaskan pada bab sebelumnya. Proses perhitungan nilai RI dapat dilihat pada Kode Sumber 4.1-2.

1	jumlah = size(A, 1);
2	RI = (1.98*(jumlah-2))/jumlah;

Kode Sumber 4.1-2 Kode Sumber Proses Perhitungan RI

4.1.3. Pembentukan Normalisasi Matriks A

Pada tahapan ini akan dilakukan proses pembentukan sebuah matriks baru yang merupakan proses normalisasi dari

matriks A . Proses pembentukan matriks baru hasil dari normalisasi matriks A dapat dilihat pada Kode Sumber 4.1-3.

1	<code>for i = 1 : jumlah</code>
2	<code> totalsum(i) = sum(A(:, i));</code>
3	<code>end</code>
4	<code>normalizedA = zeros (jumlah, jumlah);</code>
5	<code>for i = 1 : jumlah</code>
6	<code> normalizedA(:,i) = A(:,i) /</code>
7	<code> totalsum(i);</code>
8	<code>end</code>

Kode Sumber 4.1-3 Kode Sumber Pembentukan Normalisasi Matriks A

4.1.4. Hitung Bobot Vektor Matriks A

Pada tahapan ini akan dilakukan proses perhitungan nilai bobot vektor dari matriks A . Proses perhitungan nilai bobot vektor dari matriks A dapat dilihat pada Kode Sumber 4.1-4.

1	<code>for i = 1 : jumlah</code>
2	<code> U(i) = sum(normalizedA(i,:)) / jumlah;</code>
3	<code>end</code>
4	<code>U = transpose(U);</code>

Kode Sumber 4.1-4 Kode Sumber Perhitungan Bobot Vektor Matriks A

4.1.5. Perhitungan CI

Pada tahapan ini akan dilakukan proses perhitungan nilai CI . Proses perhitungan nilai CI dapat dilihat pada Kode Sumber 4.1-5.

1	<code>n = A * U;</code>
2	<code>nmax = sum(n);</code>
3	<code>CI = (nmax - jumlah) / (jumlah - 1);</code>

Kode Sumber 4.1-5 Kode Sumber Perhitungan Nilai CI

4.1.6. Evaluasi Kelayakan Matriks A

Pada tahapan ini akan dilakukan proses evaluasi kelayakan dari Matriks A. Proses evaluasi kelayakan dari matriks A dapat dilihat pada Kode Sumber 4.1-6.

1	CR = CI / RI;
2	if CR < 0.1
3	disp 'Acceptable';
4	else
5	disp 'Non Acceptable';
6	end

Kode Sumber 4.1-6 Kode Sumber Evaluasi Kelayakan dari Matriks A

4.1.7. Proses Pembentukan Matriks Turunan

Pada tahapan ini akan dilakukan proses pembentukan sejumlah kriteria matriks baru yang akan membandingkan nilai hasil implementasi dari kedua metode GA dan ACO. Proses pembentukan matriks turunan dapat dilihat pada Kode Sumber 4.1-7.

1	if CR < 0.1
2	disp 'Acceptable';
3	Awaktu = [1 0.1;
4	9 1];
5	Amemori = [1 0.1;
6	9 1];
7	Ajarak =[1 1
8	1 1];
9	Akode = [1 0.2
10	5 1];

Kode Sumber 4.1-7 Kode Sumber Proses Pembentukan Matriks Turunan

4.1.8. Pembentukan Normalisasi Matriks Turunan

Pada tahapan ini akan dilakukan proses pembentukan matriks normalisasi dari setiap matriks turunan. Proses

pembentukan matriks baru hasil dari normalisasi matrik turunan dapat dilihat pada Kode Sumber 4.1-8.

1	<code>for i = 1 : jmlahturunan</code>
2	<code>totalsum1(i) = sum (Awaktu(:,i));</code>
3	<code>totalsum2(i) = sum (Amemori(:,i));</code>
4	<code>totalsum3(i) = sum (Ajarak(:,i));</code>
5	<code>totalsum4(i) = sum (Akode(:,i));</code>
6	<code>end</code>
7	
8	<code>for i = 1 : jmlahturunan</code>
9	<code>normalizedA1(:,i) = Awaktu(:,i) /</code>
	<code>totalsum1(i);</code>
10	<code>normalizedA2(:,i) = Amemori(:,i) /</code>
	<code>totalsum2(i);</code>
11	<code>normalizedA3(:,i) = Ajarak(:,i) /</code>
	<code>totalsum3(i);</code>
12	<code>normalizedA4(:,i) = Akode(:,i) /</code>
	<code>totalsum4(i);</code>
13	<code>end</code>

Kode Sumber 4.1-8 Kode Sumber Pembentukan Normalisasi dari Matriks Turunan

4.1.9. Hitung Bobot Vektor Matriks Turunan

Pada tahapan ini akan dilakukan proses perhitungan nilai bobot vektor dari keempat matriks turunan yang telah dibuat sebelumnya. Proses perhitungan bobot vektor dari matriks turunan dapat dilihat pada Kode Sumber 4.1-9.

4.1.10. Evaluasi Nilai GA Dan ACO

Pada tahapan ini akan dilakukan proses perhitungan nilai akhir dari nilai yang didapatkan untuk algoritma GA dan nilai yang didapatkan untuk algoritma ACO. Proses evaluasi nilai GA dan ACO dapat dilihat pada Kode Sumber 4.1-10.

1	<code>for i = 1 : jumlahturunan</code>
2	<code> U1(i) = sum(normalizedA1(i,:)) /</code> <code> jumlahturunan;</code>
3	<code> U2(i) = sum(normalizedA2(i,:)) /</code> <code> jumlahturunan;</code>
4	<code> U3(i) = sum(normalizedA3(i,:)) /</code> <code> jumlahturunan;</code>
5	<code> U4(i) = sum(normalizedA4(i,:)) /</code> <code> jumlahturunan;</code>
6	<code>end</code>

Kode Sumber 4.1-9 Kode Sumber Perhitungan Bobot Matriks Turunan

1	<code>PGA =</code> <code>(U(1,1)*U1(1,1))+(U(2,1)*U2(1,1))+(U(3,1)*U</code> <code>3(1,1))+(U(4,1)*U4(1,1))</code>
2	<code>PACO =</code> <code>(U(1,1)*U1(1,2))+(U(2,1)*U2(1,2))+(U(3,1)*U</code> <code>3(1,2))+(U(4,1)*U4(1,2))</code>

Kode Sumber 4.1-10 Kode Sumber Evaluasi nilai GA dan ACO

Terakhir kita dapat mengetahui total baris kode implementasi dari metode *Pairwise Comparison Matrix* berjumlah 64.

4.2. Program Fungsi Euclidean

Pada tahap ini akan dilakukan pengolahan data masukan dengan membaca data masukan yang berupa *file.txt* dan mencari jarak antar *nodes* menggunakan persamaan Euclidean yang sudah dijelaskan pada Persamaan 3.1. Proses perhitungan fungsi Euclidean dapat dilihat pada Kode Sumber 4.2-1.

1	<code>fileID = fopen ('eil51.txt', 'r');</code>
2	<code>formatSpec = '%d %d %d';</code>
3	<code>sizeA = [3 inf];</code>
4	<code>A = fscanf(fileID, formatSpec, sizeA);</code>
5	<code>fclose (fileID);</code>

6	jumlah = size(A,2);
7	jarak = zeros(jumlah, jumlah);
8	for asal = 1 : jumlah
9	for tujuan = 1 : jumlah
10	if asal == tujuan
11	jarak (asal, tujuan) = 0;
12	else
13	jarak (asal, tujuan) =
	sqrt(abs(A(2, asal) - A(2, tujuan)).^2 +
	abs(A(3, asal) - A(3, tujuan)).^2);
14	end
15	end
16	end

Kode Sumber 4.2-1 Kode Sumber Implementasi Euclidean

4.3. Program GA.m

Pada program ini berisikan jalannya proses penyelesaian TSP menggunakan GA. Tahapan jalannya program ini adalah sebagai berikut.

4.3.1. Proses Inisialisasi Data

Pada tahapan ini akan dilakukan inisialisasi data awal, yaitu menentukan data masukan awal, data ukuran populasi, data jumlah iterasi, data nilai probabilitas untuk *crossover*, dan data nilai untuk probabilitas mutasi.

Data masukan awal merupakan data masukan dengan format *text* (*filename.txt*) yang terdiri dari 3 kolom utama yaitu, urutan *nodes*, koordinat *x*, dan koordinat *y* dari setiap *nodes*. Semua data ini akan direpresentasikan ke dalam matriks yang bernama A, yang dilakukan pada preproses dataset ditunjukkan oleh Kode Sumber 4.2-1. Data ukuran populasi ditetapkan bernilai 2000, yang berarti terdapat 2000 individu (yang setiap individu memiliki kemungkinan untuk menjadi *parent* yang melakukan *crossover*). Data jumlah iterasi merupakan nilai yang kita masukan untuk menentukan jumlah perulangan yang kita inginkan. Jumlah iterasi ditetapkan bernilai 1000, dan disimpan dalam variabel

maksiterasi. Data nilai probabilitas untuk *crossover* ditetapkan bernilai 0.5, yang berarti terdapat kemungkinan 50% *parent* akan melakukan *crossover*. Nilai probabilitas untuk *crossover* disimpan dalam variabel `pco`. Dan data nilai probabilitas untuk mutasi ditetapkan bernilai 0.01, dengan demikian setiap *child* yang dihasilkan dari proses *crossover* memiliki kemungkinan 1% untuk mengalami mutasi. Nilai probabilitas untuk mutasi disimpan dalam variabel `pm`. Proses inialisasi data untuk metode GA dapat dilihat pada Kode Sumber 4.3-1.

1	<code>maksiterasi = 1000;</code>
2	<code>populasi = 2000;</code>
3	<code>pco = 0.5;</code>
4	<code>pm = 0.01;</code>

Kode Sumber 4.3-1 Kode Sumber Inialisasi Data Metode GA

4.3.2. Proses Membangkitkan Generasi Pertama

Pada tahapan ini akan dilakukan proses pembangkitan generasi pertama. Proses ini adalah membangkitkan sejumlah kromosom untuk yang pertama kali. Pembangkitan kromosom dilakukan berdasarkan urutan *nodes* secara acak. Proses pembangkitan kromosom generasi pertama dapat dilihat pada Kode Sumber 4.3-2.

4.3.3. Proses Penentuan Operator

Pada tahapan ini akan dilakukan proses penentuan operator *crossover* dan operator mutasi. Yang dimaksud disini adalah penentuan titik *crossover* dan titik mutasi. Proses penentuan titik operator dapat dilihat pada Kode Sumber 4.3-3.

1	<code>if iter == 1</code>
2	<code>for I = 1 : populasi</code>
3	<code>temp = randperm (jumlah, jumlah);</code>
4	<code>for j = 1 : jumlah</code>
5	<code>parent (I, j) = temp (1, j);</code>
6	<code>end</code>
7	<code>end</code>
8	<code>end</code>

Kode Sumber 4.3-2 Kode Sumber Pembangkitan Generasi Pertama

1	<code>operator = randperm (jumlah - 1, 2);</code>
2	<code>c1 = min (operator);</code>
3	<code>if c1 == 1</code>
4	<code>c1 = c1 + 1;</code>
5	<code>end</code>
6	<code>c2 = max (operator);</code>
7	<code>if c2 == c1</code>
8	<code>c2 = c2 + 1;</code>
9	<code>end</code>
10	
11	<code>operanm = randperm (jumlah - 1, 2);</code>
12	<code>m1 = min (operanm);</code>
13	<code>m2 = max (operanm);</code>

Kode Sumber 4.3-3 Kode Sumber Penentuan Titik Operator

4.3.4. Proses Seleksi *Parent*

Pada tahapan ini akan dilakukan proses pemilihan *parent* yang akan mengalami kawin silang. Berdasarkan yang telah dijelaskan pada bab sebelumnya. Awalnya setiap calon *parent* akan dibangkitkan sebuah bilangan acak untuk dibandingkan dengan probabilitas *crossover* yang sudah ditentukan sebelumnya. Proses seleksi *parent* dapat dilihat pada Kode Sumber 4.3-4.

1	<code>for I = 1 : populasi</code>
2	<code> parentpCO (i) = rand;</code>
3	<code>end</code>
4	
5	<code>availCO = find (parentpCO < pco);</code>
6	<code>iterCO = size(availCO, 2) - 1;</code>

Kode Sumber 4.3-4 Kode Sumber Proses Seleksi Parent

4.3.5. Proses Crossover

Pada tahapan ini akan dilakukan proses kawin silang. Proses kawin silang dilakukan dengan membutuhkan dua *parent* yang sudah ditentukan pada proses sebelumnya. Urutan kawin silang yang akan dilakukan yaitu $parent_i$ dengan $parent_{i+1}$ sampai dengan $parent_{n-1}$ dengan $parent_n$. Proses kawin silang membutuhkan beberapa sub-proses yaitu:

4.3.5.1. Proses Penukaran Posisi Operator

Pada sub-proses ini akan dilakukan proses penukaran posisi titik operator seperti yang sudah dijelaskan pada Bab 3. Pada proses ini juga mulai terjadi pembentukan *child* baru dengan nilai yang belum sempurna. Yang mana hasil penukaran posisi titik operator yang berasal dari *parent* disimpan kedalam *child*. Proses penukaran posisi titik operator *crossover* dapat dilihat pada Kode Sumber 4.3-5.

1	<code>for I = 1 : iterCO</code>
2	<code> for j = c1 : c2</code>
3	<code> child (I + I, j) = parent</code>
	<code> (availCO(i) , j);</code>
4	<code> child (I + I - 1, j) = parent</code>
	<code> (availCO(I + 1), j);</code>
5	<code> end</code>
6	<code>end</code>

Kode Sumber 4.3-5 Kode Sumber Proses Penukaran Posisi Titik Operator

4.3.5.2. Proses Pembentukan L

Pada sub-proses ini akan dilakukan proses pembentukan L seperti yang sudah dijelaskan pada Bab 3. *L* merupakan list yang berisi dengan elemen-elemen yang berasal dari *parent* dengan prosedur pengisian dengan urutan $c_2 + 1, c_2 + 2, \dots, n, 1, 2, \dots, c_2$ atau bisa dibilang searah dengan jarum jam. Proses pembentukan list *L* dalam proses *crossover* dapat dilihat pada Kode Sumber 4.3-6.

```

1  for i = 1 : iterCO
2      tempc = c2 + 1;
3      for j = 1 : jumlah - c2
4          L (i + i - 1, j) = parent
      (availCO(i) , tempc);
5          L (i + i, j) = parent (availCO(i +
      1), tempc);
6          pos = j;
7          tempc = tempc + 1;
8      end
9  end
10 tempc = pos + c2;
11 pos = pos + 1;
12
13 for i = 1 : iterCO
14     abi = 1;
15     for j = pos : tempc
16         L (i + i - 1, j) = parent
      (availCO(i) , abi);
17         L (i + i, j) = parent (availCO(i +
      1), abi);
18         abi = abi + 1;
19     end
20 end

```

Kode Sumber 4.3-6 Kode Sumber Proses Pembentukan List L

4.3.5.3. Proses Pembentukan L'

Pada sub-proses ini akan dilakukan proses pembentukan L' seperti yang sudah dijelaskan pada Bab 3. L' merupakan list yang berisi elemen-elemen berdasarkan list L yang sudah dihapus dengan elemen-elemen yang sudah terdaftar pada *child* sementara. Dengan kata lain L' dapat dikatakan list yang berisi list L yang berisikan dengan list *child*. Pada program fungsi tes, merupakan fungsi yang berfungsi untuk mengambil nilai irisan. Proses pembentukan list L' dapat dilihat pada Kode Sumber 4.3-7. Sementara proses untuk mendapatkan elemen dari irisan list *child* sementara dapat dilihat pada Kode Sumber 4.3-8.

1	Laksen = zeros (iterCO * 2, jumlah);
2	
3	for i = 1 : iterCO
4	tmp1 = tes(L(i + i - 1,:), child(i + i
5	- 1,:));
6	tmp2 = tes(L(i + i,:), child(i +
7	i,:));
8	for j = 1 : jumlah - (c2 - c1 + 1)
9	Laksen (i + i - 1, j) = tmp1 (1,
10	j);
	Laksen (i + i, j) = tmp2 (1, j);
	end
	end

Kode Sumber 4.3-7 Kode Sumber Proses Pembentukan List L'

1	function y = tes (L, child)
2	[a b] = setxor (L, child);
3	c = sort(b);
4	for i = 1 : size (c,2)
5	temp = c (i);
6	y (i) = L (temp);
7	end
8	end

Kode Sumber 4.3-8 Kode Sumber Proses Mencari Irisan List L dengan Child

4.3.5.4. Proses Pembentukan Child

Pada sub-proses ini akan dilakukan proses pembentukan child seperti yang sudah dijelaskan pada Bab 3. Ini merupakan lanjutan dari tahap 1 proses *crossover*. Dimana pada tahap sebelumnya *child* sudah memiliki gen sejumlah selisih titik *crossover* yang telah ditentukan sebelumnya. Dan pada proses ini sendiri dilanjutkan proses pengisian pada *child* tersebut. Proses pembentukan *child* dapat dilihat pada Kode Sumber 4.3-9.

```

1  for I = 1 : iterCO
2      counter (I + I - 1) = 1;
3      counter (I + i) = 1;
4      for j = c2 + 1 : jumlah
5          child (I + I - 1, j) = Laksen (I +
I - 1, counter (I + I - 1));
6          child (I + I, j) = Laksen (I + I,
counter (I + i));
7          counter (I + I - 1) = counter (I +
I - 1) + 1;
8          counter (I + i) = counter (I + i)
+ 1;
9      end
10 end
11
12 for I = 1 : iterCO
13     for j = 1 : c1 - 1
14         child (I + I - 1, j) = Laksen (I +
I - 1, counter (I + I - 1));
15         child (I + I, j) = Laksen (I + I,
counter (I + i));
16         counter (I + I - 1) = counter (I +
I - 1) + 1;
17         counter (I + i) = counter (I + i)
+ 1;
18     end
19 end

```

Kode Sumber 4.3-9 Kode Sumber Proses Pembentukan Child

4.3.6. Proses Mutasi

Pada tahapan ini akan dilakukan proses mutasi. Proses mutasi terjadi hanya pada *child* yang terpilih dengan nilai probabilitasnya lebih rendah dari yang sudah ditentukan. Proses mutasi akan memberikan individu yang baru dan berbeda dari proses *crossover*. Probabilitas mutasi ini sendiri sudah ditentukan bernilai 0.01. Proses mutasi dapat dilihat pada Kode Sumber 4.3-10.

1	<code>for I = 1 : size (child, 1)</code>
2	<code> childPM(i) = rand;</code>
3	<code>end</code>
4	
5	<code>if size (child, 1) > 0</code>
6	<code> availM = find (childPM < pm);</code>
7	<code> for I = 1 : size (availM, 2)</code>
8	<code> mutasitemp = child (availM(i),</code>
9	<code> m1);</code>
10	<code> child (availM(i), m1) = child</code>
11	<code> (availM(i), m2);</code>
12	<code> child (availM(i), m2) =</code>
	<code> mutasitemp;</code>
	<code> end</code>
	<code>end</code>

Kode Sumber 4.3-10 Kode Sumber Proses Mutasi

4.3.7. Proses Evaluasi Fitness

Pada tahapan ini akan dilakukan proses perhitungan fitness dari setiap kromosom dilihat dari gen masing-masing untuk mendapatkan jarak tempuh total. Dengan catatan setiap kromosom mendapatkan gen tambahan yang berupa sama dengan gen pertama pada kromosom tersebut. Proses evaluasi nilai *fitness* setiap kromosom dapat dilihat pada Kode Sumber 4.3-11.

```

1  x = 1;
2  for i = jumpopul : populasi + size (child,
   1)
3      individu (i,:) = child (x,:);
4      x = x + 1;
5  end
6
7  for i = 1 : size (individu, 1)
8      jaraksementara = 0;
9      for j = 1 : size (individu, 2)
10         if j == jumlah
11             jaraksementara =
jaraksementara + jarak(individu(i, j),
individu(i, 1));
12         else
13             jaraksementara =
jaraksementara + jarak(individu(i, j),
individu(i, j + 1));
14         end
15     end
16     fitness (i) = jaraksementara;
17 end

```

Kode Sumber 4.3-11 Kode Sumber Proses Evaluasi Fitness

4.3.8. Proses Elitisme dan Penentuan Jalur

Pada tahapan ini akan dilakukan proses elitism, yaitu perbaikan kromosom untuk generasi berikutnya. Dan untuk perulangan terakhir dilakukan penentuan jalur terbaik. Proses Elitisme ini dilakukan dengan cara mengurutkan setiap kromosom berdasarkan nilai *fitness*-nya. Sejumlah populasi kromosom yang diinginkan dengan nilai *fitness* terbaik menjadi populasi terpilih untuk generasi berikutnya. Terakhir proses Elitisme dan penentuan jalur terpendek dapat dilihat pada Kode Sumber 4.3-12.

1	[sortedfitness urutan] = sort (fitness);
2	for i = 1 : populasi
3	for j = 1 : jumlah
4	parent (i, j) =
	individu(urutan(i), j);
5	end
6	end

Kode Sumber 4.3-12 Kode Sumber Proses Elitisme

Terakhir kita dapat mengetahui total baris kode implementasi dari metode GA berjumlah 261.

4.4. Program ACO.m

4.4.1. Proses Inisialisasi Data

Pada tahapan ini akan dilakukan inisialisasi data awal, yaitu menentukan data masukan awal, data jumlah iterasi, data jumlah semut, data koefisien nilai evaporasi, data koefisien nilai eliminasi, data nilai alpha, dan data nilai beta.

Data masukan awal merupakan data masukan dengan format Text (*filename.txt*) yang terdiri dari 3 kolom utama yaitu, urutan *nodes*, koordinat x , dan koordinat y dari setiap *nodes*. Semua data ini akan direpresentasikan ke dalam matriks yang bernama A, yang dilakukan pada preproses dataset ditunjukkan oleh Kode Sumber 4.2-1. Data jumlah iterasi merupakan nilai yang kita masukan untuk menentukan jumlah perulangan yang kita inginkan. Jumlah iterasi ditetapkan bernilai 100, dan disimpan dalam variabel *maksiterasi*. Data jumlah semut merupakan nilai yang kita masukan untuk menentukan jumlah semut yang ingin kita bangkitkan. Jumlah semut ditentukan bernilai 200, dan disimpan dalam variabel *m*. Data koefisien nilai evaporasi adalah nilai koefisien untuk evaporasi. Yaitu saat pembaharuan nilai *pheromones* seperti yang dijelaskan pada bab sebelumnya ketika suatu jalur tidak dilewati semut. Koefisien evaporasi ditetapkan bernilai 0.1, dan disimpan

dalam variabel `eva`. Data koefisien nilai eliminasi adalah nilai yang dibutuhkan pada proses eliminasi. Data koefisien eliminasi ditetapkan bernilai 0.96, dan disimpan dalam variabel `eli`. Data nilai alpha adalah nilai koefisien dari α yang berarti urutan kemungkinan *next state* dalam perjalanan semut. Koefisien alpha ditentukan bernilai 1, dan disimpan dalam variabel `alpha`. Data nilai beta adalah nilai koefisien dari beta yang berarti urutan *state* yang telah dilalui seekor semut. Koefisien beta ditentukan bernilai 5, dan disimpan dalam variabel `beta`. Proses untuk inialisasi data metode ACO dapat dilihat pada Kode Sumber 4.4-1.

1	<code>maksiterasi = 100;</code>
2	<code>m = 100;</code>
3	<code>eva = 0.1;</code>
4	<code>eli = 0.1;</code>
5	<code>alpha = 1;</code>
6	<code>beta = 5;</code>

Kode Sumber 4.4-1 Kode Sumber Inialisasi Data Metode ACO

4.4.2. Proses Penempatan Semut

Pada tahapan ini akan dilakukan proses penempatan semut, yaitu proses dimana seekor semut akan memiliki *entry state* tempat memulai dan mengakhiri perjalanan masing-masing semut. Proses ini dilakukan dengan membangkitkan sebuah bilangan acak untuk masing-masing semut. Dan hasil pembangkitan tersebut disimpan pada sebuah list yang akan melakukan pencatatan jalur yang ditempuh seekor semut. Proses penempatan semut dapat dilihat pada Kode Sumber 4.4-2.

1	<code>randnodes = randi(jumlah, 1, m);</code>
2	<code>for i = 1 : m</code>
3	<code> antpos (i, 1) = randnodes(1, i);</code>
4	<code>end</code>

Kode Sumber 4.4-2 Kode Sumber Proses Penempatan Semut

4.4.3. Proses Perjalanan Semut

Pada tahapan ini akan dilakukan proses perjalanan semut, yaitu proses menentukan tujuan berikutnya pada suatu semut jika semut tersebut masih dalam perjalanan atau belum sampai pada tujuan (kota asal). Seekor semut akan memiliki probabilitas yang sudah dijelaskan pada Bab 3. Dimana jalur yang memiliki kemungkinan lebih besar terpilih adalah jalur dengan nilai *pheromones* yang lebih besar. Proses perjalanan semut dapat dilihat pada Kode Sumber 4.4-3.

```

1  for i = 1 : m
2      mpherom = pheromones;
3      for j = 1 : jumlah - 1
4          posisi = antpos (i, j);
5          mpherom (:, posisi) = 0;
6
7          temp =
(t(posisi,:) .^beta) .*(mpherom(posisi,:) .^a
lpha);
8          s = sum(temp);
9          probabilitas = temp/s;
10
11         s = 0;
12         r = rand;
13
14         for k = 1 : jumlah
15             s = s + probabilitas(k);
16             if r < s
17                 antpos (i, j + 1) = k;
18                 break
19             end
20         end
21     end
22 end
23 anttour = antpos;
24 anttour = horzcat(anttour, anttour(:, 1));

```

Kode Sumber 4.4-3 Kode Sumber Proses Penentuan Jalur Semut

4.4.4. Proses Perhitungan *Cost*

Pada tahapan ini akan dilakukan proses perhitungan *cost*, yaitu proses menghitung total jarak tempuh pada jalur yang dilalui seekor semut untuk menyelesaikan TSP. Proses perhitungan *cost* dapat dilihat pada Kode Sumber 4.4-4.

1	<code>for i = 1 : m</code>
2	<code> s = 0;</code>
3	<code> for j = 1 : jumlah</code>
4	<code> s = s + distance (anttour(i, j),</code> <code> anttour(i, j + 1));</code>
5	<code> end</code>
6	<code> fitness (i) = s;</code>
7	<code>end</code>
8	<code>cost = fitness;</code>
9	<code>fitness = fitness - eli * min(fitness);</code>

Kode Sumber 4.4-4 Kode Sumber Proses Perhitungan *Cost*

4.4.5. Proses Pembaharuan Nilai *Pheromones*

Pada tahapan ini akan dilakukan pembaharuan nilai *pheromones*, yaitu proses memperbarui nilai *pheromones*. Nilai *pheromones* akan mengalami peningkatan pada jalur yang dilewati semut, sebaliknya *pheromones* akan mengalami penguapan jika jalur tersebut tidak dilewati semut. Proses pembaharuan nilai *pheromones* dapat dilihat pada Kode Sumber 4.4-5.

1	<code>for i = 1 : m</code>
2	<code> for j = 1 : jumlah</code>
3	<code> dt = 1 / cost (i);</code>
4	<code> t(anttour(i, j), anttour(i, j + 1))</code> <code>= (1 - eva) * t(anttour(i, j), anttour(i,</code> <code> j + 1)) + dt;</code>
5	<code> end</code>
6	<code>end</code>

Kode Sumber 4.4-5 Kode Sumber Proses Pembaharuan *Pheromones*

4.4.6. Proses Penentuan Jalur Terpendek

Pada tahapan ini akan dilakukan proses penentuan jalur terpendek, yaitu proses menentukan sebuah jalur dari beberapa calon jalur yang dibuat oleh semut. Jalur dengan nilai *cost* terendah ditetapkan sebagai jalur terpendek. Terakhir proses penentuan jalur terpendek dapat dilihat pada Kode Sumber 4.4-6.

1	<code>terpendek (iter) = min(cost);</code>
2	<code>palingpendek = min(terpendek)</code>

Kode Sumber 4.4-6 Kode Sumber Proses Penentuan Jalur Terpendek

Terakhir kita dapat mengetahui total baris kode implementasi dari metode ACO berjumlah 152.

(halaman ini sengaja dikosongkan)

BAB V

UJICOBADA DAN EVALUASI

5.1.Lingkungan Pelaksanaan Uji Coba

Lingkungan uji coba yang digunakan dalam pembuatan Tugas Akhir ini meliputi perangkat lunak dan perangkat keras yang digunakan untuk membandingkan metode kecerdasan buatan untuk mendapatkan metode yang lebih baik dalam menyelesaikan persoalan TSP. Lingkungan uji coba merupakan komputer tempat uji coba perangkat lunak. Berikut adalah lingkungan uji coba yang digunakan pada Tugas Akhir ini seperti yang ditampilkan pada Tabel 5.1-1.

Tabel 5.1-1 Lingkungan Uji Coba Perangkat Lunak

Perangkat	Spesifikasi	
Perangkat Keras	Prosesor	Intel(R) Core (TM) : i5 CPU 3210M @2.50 GHz
	Memori	: 4.0 GB
Perangkat Lunak	Sistem Operasi	Windows 7 Basic : 32-bit
	Perangkat Pengembang	: Matlab R2011b,

5.2.Data Uji Coba

Data yang digunakan pada uji coba ini adalah sebuah grafik yang setiap *nodes*-nya terhubung satu sama lain (*fully connected graph*). Tujuan utama dari data adalah mencari perbedaan hasil akhir dari implementasi kedua algoritma kecerdasan buatan yang telah disebutkan sebelumnya.

5.3. Skenario Uji Coba

Pada bagian ini akan dijelaskan skenario uji coba program optimasi yang sudah dikembangkan. Skenario dalam uji coba ini adalah uji coba pengubahan jumlah *nodes* data masukan. Tujuan dari uji coba melakukan pengubahan jumlah *nodes* data masukan adalah untuk mengetahui apakah jumlah *nodes* memberikan dampak pada keempat *varians* yang dibandingkan.

Pada uji coba metode *Pairwise Comparison Matrix* ini akan dilakukan skenario pengubahan data masukan awal. Perubahan yang terjadi adalah pada jumlah *nodes* awal. Terdapat 6 data masukan yang terdiri dari 14 *nodes*, 16 *nodes*, 22 *nodes*, 51 *nodes*, 52 *nodes* dan x *nodes*. Setiap skenario akan diuji coba sebanyak 5 kali.

5.3.1. Uji Coba 14 Nodes

Pada uji coba ini data masukan awal berjumlah 14 *nodes*. Hasil percobaan penyelesaian TSP menggunakan metode GA dapat dilihat pada Tabel 5.3-1. Sementara hasil percobaan penyelesaian TSP menggunakan metode ACO dapat dilihat pada Tabel 5.3-2. Pada uji coba kali ini nilai target berasal dari <http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/stsp-sol.html>. Hasil jalur tempuh terbaik dari uji coba ditunjukkan dengan Gambar A 7 untuk metode GA dan Gambar A 13 untuk metode ACO.

5.3.2. Uji Coba 16 Nodes

Pada uji coba ini data masukan awal berjumlah 16 *nodes*. Hasil percobaan penyelesaian TSP menggunakan metode GA dapat dilihat pada Tabel 5.3-3. Sementara hasil percobaan penyelesaian TSP menggunakan metode ACO dapat dilihat pada Tabel 5.3-4. Pada uji coba kali ini nilai target berasal dari <http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/stsp-sol.html>. Hasil jalur tempuh terbaik

dari uji coba ditunjukkan dengan Gambar A 8 untuk metode GA dan Gambar A 14 untuk metode ACO.

Tabel 5.3-1 Hasil 5 Percobaan Uji Coba 14 Nodes dengan Metode GA

<i>Dataset</i>	<i>burma14</i>	
n	14	
<i>Target</i>	3323	
Percobaan	Hasil	Akurasi
1	3087.85	7.08% lebih baik
2	3087.85	7.08% lebih baik
3	3087.85	7.08% lebih baik
4	3087.85	7.08% lebih baik
5	3087.85	7.08% lebih baik
<i>Mean</i>	3087.85	7.08% lebih baik
Waktu	414.27s	
Memori	45.3 MB	

Tabel 5.3-2 Hasil 5 Percobaan Uji Coba 14 Nodes dengan Metode ACO

<i>Dataset</i>	<i>burma14</i>	
n	14	
<i>Target</i>	3323	
Percobaan	Hasil	Akurasi
1	3087.85	7.08% lebih baik
2	3087.85	7.08% lebih baik
3	3087.85	7.08% lebih baik
4	3087.85	7.08% lebih baik
5	3087.85	7.08% lebih baik
<i>Mean</i>	3087.85	7.08% lebih baik
Waktu	3.22s	
Memori	1.35 MB	

Tabel 5.3-3 Hasil 5 Percobaan Uji Coba 16 Nodes dengan Metode GA

<i>Dataset</i>	<i>ulysses16</i>	
n	16	
<i>Target</i>	6859	
Percobaan	Hasil	Akurasi
1	7398.76	92.13%
2	7419.89	91.82%
3	7399.98	92.11%
4	7399.98	92.11%
5	7398.76	92.13%
<i>Mean</i>	7403.48	92.06%
Waktu	385.06s	
Memori	45.9 MB	

Tabel 5.3-4 Hasil 5 Percobaan Uji Coba 16 Nodes dengan Metode ACO

<i>Dataset</i>	<i>ulysses16</i>	
n	16	
<i>Target</i>	6859	
Percobaan	Hasil	Akurasi
1	7400.13	92.11%
2	7398.76	92.13%
3	7398.76	92.13%
4	7400.13	92.11%
5	7398.76	92.13%
<i>Mean</i>	7399.31	92.12%
Waktu	4.091s	
Memori	1.94 MB	

5.3.3. Uji Coba 22 Nodes

Pada uji coba ini data masukan awal berjumlah 22 *nodes*. Hasil percobaan penyelesaian TSP menggunakan metode GA dapat dilihat pada Tabel 5.3-5. Sementara hasil percobaan penyelesaian TSP menggunakan metode ACO dapat dilihat pada Tabel 5.3-6. Pada uji coba kali ini nilai target berasal dari <http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/stsp-sol.html>. Hasil jalur tempuh terbaik dari uji coba ditunjukkan dengan Gambar A 9 untuk metode GA dan Gambar A 15 untuk metode ACO.

5.3.4. Uji Coba 51 Nodes

Pada uji coba ini data masukan awal berjumlah 51 *nodes*. Hasil percobaan penyelesaian TSP menggunakan metode GA dapat dilihat pada Tabel 5.3-7. Sementara hasil percobaan penyelesaian TSP menggunakan metode ACO dapat dilihat pada Tabel 5.3-8. Pada uji coba kali ini nilai target berasal dari <http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/stsp-sol.html>. Hasil jalur tempuh terbaik dari uji coba ditunjukkan dengan Gambar A 10 untuk metode GA dan Gambar A 16 untuk metode ACO.

5.3.5. Uji Coba 52 Nodes

Pada uji coba ini data masukan awal berjumlah 52 *nodes*. Hasil percobaan penyelesaian TSP menggunakan metode GA dapat dilihat pada Tabel 5.3-9. Sementara hasil percobaan penyelesaian TSP menggunakan metode ACO dapat dilihat pada Tabel 5.3-10. Pada uji coba kali ini nilai target berasal dari <http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/stsp-sol.html>. Hasil jalur tempuh terbaik dari uji coba ditunjukkan dengan Gambar A 11 untuk metode GA dan Gambar A 17 untuk metode ACO.

Tabel 5.3-5 Hasil 5 Percobaan Uji Coba 22 *Nodes* dengan Metode GA

<i>Dataset</i>	<i>ulysses22</i>	
n	22	
<i>Target</i>	7013	
Percobaan	Hasil	Akurasi
1	7530.97	92.61%
2	7644.76	90.99%
3	7643.58	91.01%
4	7588.05	91.80%
5	7701.26	90.19%
<i>Mean</i>	7621.73	91.32%
Waktu	500.91s	
Memori	50.9 MB	

Tabel 5.3-6 Hasil 5 Percobaan Uji Coba 22 *Nodes* dengan Metode ACO

<i>Dataset</i>	<i>ulysses22</i>	
n	22	
<i>Target</i>	7013	
Percobaan	Hasil	Akurasi
1	7626.34	91.25%
2	7667.51	90.67%
3	7659.08	90.79%
4	7820.91	88.48%
5	7601.59	91.61%
<i>Mean</i>	7675.08	90.56%
Waktu	6.10s	
Memori	2.61 MB	

Tabel 5.3-7 Hasil 5 Percobaan Uji Coba 51 Nodes dengan Metode GA

<i>Dataset</i>	<i>eil51</i>	
N	51	
<i>Target</i>	426	
Percobaan	Hasil	Akurasi
1	504.88	81.48%
2	466.40	90.52%
3	522.74	77.29%
4	518.83	78.21%
5	487.94	85.46%
<i>Mean</i>	500.16	82.59%
Waktu	1536.62s	
Memori	82.8 MB	

Tabel 5.3-8 Hasil 5 Percobaan Uji Coba 51 Nodes dengan Metode ACO

<i>Dataset</i>	<i>eil51</i>	
n	51	
<i>Target</i>	426	
Percobaan	Hasil	Akurasi
1	522.36	77.38%
2	545.48	71.95%
3	540.32	73.17%
4	530.38	75.50%
5	501.35	82.31%
<i>Mean</i>	527.98	76.06%
Waktu	19.35s	
Memori	10.1 MB	

Tabel 5.3-9 Hasil 5 Percobaan Uji Coba 52 Nodes dengan Metode GA

Dataset	<i>berlin52</i>	
N	52	
Target	7542	
Percobaan	Hasil	Akurasi
1	9277.34	76.99%
2	8837.28	82.83%
3	8607.82	85.87%
4	9420.48	75.09%
5	8795.72	83.38%
Mean	8987.73	80.83%
Waktu	1500.31s	
Memori	98.3 MB	

Tabel 5.3-10 Hasil 5 Percobaan Uji Coba 52 Nodes dengan Metode ACO

Dataset	<i>berlin52</i>	
n	52	
Target	7542	
Percobaan	Hasil	Akurasi
1	9460.08	74.57%
2	9275.86	77.01%
3	9321.33	76.41%
4	9629.70	72.32%
5	9381.95	75.60%
Mean	9413.78	75.18%
Waktu	19.99s	
Memori	11.3 MB	

5.3.6. Uji Coba 76 nodes

Pada uji coba ini data masukan awal berjumlah 76 nodes. Hasil percobaan penyelesaian TSP menggunakan metode GA dapat dilihat pada Tabel 5.3-11. Sementara hasil percobaan penyelesaian TSP menggunakan metode ACO dapat dilihat pada Tabel 5.3-12. Pada uji coba kali ini nilai target berasal dari <http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/stsp-sol.html>. Hasil jalur tempuh terbaik dari uji coba ditunjukkan dengan Gambar A 12 untuk metode GA dan Gambar A 18 untuk metode ACO.

Tabel 5.3-11 Hasil 5 Percobaan Uji Coba 76 Nodes dengan Metode GA

Dataset	<i>eil76</i>	
N	76	
Target	538	
Percobaan	Hasil	Akurasi
1	798.19	51.63%
2	891.35	34.32%
3	785.64	53.97%
4	872.72	37.74%
5	767.94	57.25%
Mean	823.17	47.0%
Waktu	1096.47s	
Memori	105 MB	

Tabel 5.3-12 Hasil 5 Percobaan Uji Coba 76 Nodes dengan Metode ACO

Dataset	<i>eil76</i>	
N	76	
Target	538	
Percobaan	Hasil	Akurasi
1	764.69	57.86%
2	736.08	63.18%
3	757.63	59.17%
4	759.59	58.81%
5	740.89	62.28%
Mean	751.77	60.26%
Waktu	38.66s	
Memori	20.6 MB	

5.4. Uji Coba Pairwise Comparison

Pada tahap ini akan dilakukan uji coba pairwise comparison dengan membandingkan dua metode yang telah di implementasikan.

5.4.1. Pembentukan Matriks A

Hasil pembentukan akhir dari matriks A berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-1.

Tabel 5.4-1 Tabel Pembentukan Matrix A

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
A	$\begin{pmatrix} 1.0000 & 5.0000 & 0.3333 & 7.0000 \\ 0.2000 & 1.0000 & 0.1428 & 3.0000 \\ 3.0000 & 7.0000 & 1.0000 & 9.0000 \\ 0.1428 & 0.3333 & 0.1111 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1.00 & 5.00 & 0.33 & 7.00 \\ 0.20 & 1.00 & 0.14 & 3.00 \\ 3.00 & 7.00 & 1.00 & 9.00 \\ 0.14 & 0.33 & 0.11 & 1.00 \end{pmatrix}$

Dengan hasil normalisasi matriks A berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-2.

Tabel 5.4-2 Tabel Pembentukan Normalisasi $Matrix A$

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
N_A	$\begin{pmatrix} 0.2303 & 0.3750 & 0.2100 & 0.3500 \\ 0.0461 & 0.0750 & 0.0900 & 0.1500 \\ 0.6908 & 0.5250 & 0.6300 & 0.4500 \\ 0.0329 & 0.0250 & 0.0700 & 0.0500 \end{pmatrix}$	$\begin{pmatrix} 0.23 & 0.375 & 0.21 & 0.35 \\ 0.04 & 0.075 & 0.09 & 0.15 \\ 0.70 & 0.525 & 0.63 & 0.45 \\ 0.03 & 0.025 & 0.07 & 0.05 \end{pmatrix}$

Dengan bobot vektor matriks A berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-3.

Tabel 5.4-3 Tabel Hasil Perhitungan Bobot Vektor $Matrix A$

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
UA	$\begin{pmatrix} 0.2913; \\ 0.903; \\ 0.5740; \\ 0.0445 \end{pmatrix}^Z$	$(0.29; 0.09; 0.57; 0.04)^Z$

Dengan n_{max} matriks A berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-4.

Tabel 5.4-4 Tabel Hasil Perhitungan n_{max} $Matrix A$

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
n_{max}	4.2689	4.27

Dengan nilai CI matriks A berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-5.

Tabel 5.4-5 Tabel Hasil Perhitungan Nilai *CI* Matrix A
Perhitungan program *Perhitungan manual*

<i>CI</i>	0.0896	$\frac{n_{max} - n}{n - 1} = 0.09$
-----------	--------	------------------------------------

Dengan nilai *RI* yang formulanya sudah ditetapkan pada bab sebelumnya. Hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-6.

Tabel 5.4-6 Tabel Hasil Perhitungan Nilai *RI* Matrix A
Perhitungan program *Perhitungan manual*

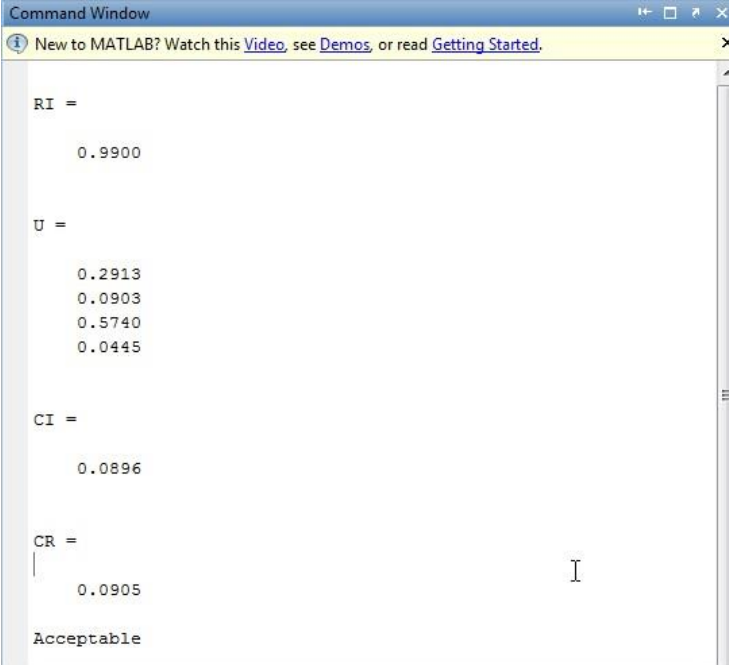
<i>RI</i>	0.9900	$\frac{1.98(n - 2)}{n} = 0.99$
-----------	--------	--------------------------------

Dengan nilai *CR* matriks A berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-7.

Tabel 5.4-7 Tabel Hasil Perhitungan Nilai *CR* Matrix A
Perhitungan program *Perhitungan manual*

<i>CR</i>	0.0905	$\frac{CI}{RI} = 0.09$
-----------	--------	------------------------

Didapatkan nilai $CR < 0.1$ yang berarti keseimbangan dari matriks A dapat diterima. Hasil pembuktian keseimbangan pada matriks A dapat dilihat pada Gambar 5.4-1.



```

Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

RI =

    0.9900

U =

    0.2913
    0.0903
    0.5740
    0.0445

CI =

    0.0896

CR =

    0.0905

Acceptable
  
```

Gambar 5.4-1 Hasil Keseimbangan Matriks A

5.4.2. Pembentukan Matriks Uji Coba 14 nodes

Hasil akhir pembentukan matriks turunan pada uji coba 14 *nodes* berdasarkan perhitungan hasil komputasi program dan hasil komputasi manual yang sudah dilakukan pada Bab 3 dapat dilihat pada Tabel 5.4-8.

Dengan hasil normalisasi matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-9.

Dan didapatkan bobot vektor dari matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-10.

Tabel 5.4-8 Tabel *Matrix* Turunan Uji Coba 14 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
A_{waktu}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{memori}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{jarak}	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$
A_{kode}	$\begin{pmatrix} 1.0000 & 0.3300 \\ 3.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}$

Tabel 5.4-9 Tabel Normalisasi *Matrix* Uji Coba 14 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
$N_{A_{waktu}}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{memori}}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{jarak}}$	$\begin{pmatrix} 0.5000 & 0.5000 \\ 0.5000 & 0.5000 \end{pmatrix}$	$\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$
$N_{A_{kode}}$	$\begin{pmatrix} 0.2500 & 0.2481 \\ 0.7500 & 0.7519 \end{pmatrix}$	$\begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}$

Tabel 5.4-10 Tabel Nilai Bobot Vektor Uji Coba 14 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
UA_{waktu}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{memori}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{jarak}	$(0.5000; 0.5000)^Z$	$(0.5; 0.5)^Z$
UA_{kode}	$(0.2491; 0.7509)^Z$	$(0.25; 0.75)^Z$

Didapatkan hasil nilai akhir untuk metode GA dan ACO berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-11.

Tabel 5.4-11 Tabel Nilai Akhir P Uji Coba 14 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
$P(GA)$	0.3345	0.334
$P(ACO)$	0.6655	0.666

5.4.3. Pembentukan Matriks Uji Coba 16 nodes

Hasil akhir pembentukan matriks turunan pada uji coba 16 nodes berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-12.

Tabel 5.4-12 Tabel *Matrix* Turunan Uji Coba 16 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
A_{waktu}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{memori}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{jarak}	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$
A_{kode}	$\begin{pmatrix} 1.0000 & 0.3300 \\ 3.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}$

Dengan hasil normalisasi matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-13.

Dan didapatkan bobot vektor dari matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-14.

Didapatkan hasil nilai akhir untuk metode GA dan ACO berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-15.

Tabel 5.4-13 Tabel Normalisasi *Matrix Uji Coba 16 Nodes*

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
N_{Awaktu}	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{Amemori}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
N_{Ajarak}	$\begin{pmatrix} 0.5000 & 0.5000 \\ 0.5000 & 0.5000 \end{pmatrix}$	$\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$
N_{Akode}	$\begin{pmatrix} 0.2500 & 0.2481 \\ 0.7500 & 0.7519 \end{pmatrix}$	$\begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}$

Tabel 5.4-14 Tabel Nilai Bobot Vektor Uji Coba 16 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
UA_{Waktu}	$(0.0905; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{memori}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{jarak}	$(0.5000; 0.5000)^Z$	$(0.5; 0.5)^Z$
UA_{kode}	$(0.2491; 0.7509)^Z$	$(0.25; 0.75)^Z$

Tabel 5.4-15 Tabel Nilai Akhir P Uji Coba 16 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
$P(GA)$	0.3345	0.334
$P(ACO)$	0.6655	0.666

5.4.4. Pembentukan Matriks Uji Coba 22 nodes

Hasil akhir pembentukan matriks turunan pada uji coba 22 nodes berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-16.

Tabel 5.4-16 Tabel Matrix Turunan Uji Coba 22 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
A_{waktu}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{memori}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{jarak}	$\begin{pmatrix} 1.0000 & 3.0000 \\ 0.3300 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 3 \\ 0.33 & 1 \end{pmatrix}$
A_{kode}	$\begin{pmatrix} 1.0000 & 0.3300 \\ 3.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}$

Dengan hasil normalisasi matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-17.

Tabel 5.4-17 Tabel Normalisasi Matrix Uji Coba 22 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
$N_{A_{waktu}}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{memori}}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{jarak}}$	$\begin{pmatrix} 0.7519 & 0.7500 \\ 0.2481 & 0.2500 \end{pmatrix}$	$\begin{pmatrix} 0.75 & 0.75 \\ 0.25 & 0.25 \end{pmatrix}$
$N_{A_{kode}}$	$\begin{pmatrix} 0.2500 & 0.2481 \\ 0.7500 & 0.7519 \end{pmatrix}$	$\begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}$

Dan didapatkan bobot vektor dari matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-18.

Tabel 5.4-18 Tabel Nilai Bobot Vektor Uji Coba 22 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
UA_{waktu}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{memori}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{jarak}	$(0.7509; 0.2491)^Z$	$(0.75; 0.25)^Z$
UA_{kode}	$(0.2491; 0.7509)^Z$	$(0.25; 0.75)^Z$

Didapatkan hasil nilai akhir untuk metode GA dan ACO berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-19.

Tabel 5.4-19 Tabel Nilai Akhir *P* Uji Coba 22 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
<i>P(GA)</i>	0.4785	0.4765
<i>P(ACO)</i>	0.5215	0.5235

5.4.5. Pembentukan Matriks Uji Coba 51 nodes

Hasil akhir pembentukan matriks turunan pada uji coba 51 nodes berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-20.

Tabel 5.4-20 Tabel *Matrix* Turunan Uji Coba 51 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
<i>A_{Waktu}</i>	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
<i>A_{memori}</i>	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
<i>A_{jarak}</i>	$\begin{pmatrix} 1.0000 & 3.0000 \\ 0.3300 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 3 \\ 0.33 & 1 \end{pmatrix}$
<i>A_{kode}</i>	$\begin{pmatrix} 1.0000 & 0.3300 \\ 3.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}$

Dengan hasil normalisasi matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-21.

Tabel 5.4-21 Tabel Normalisasi Matrix Uji Coba 51 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
N_{Awaktu}	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{memori}}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{jarak}}$	$\begin{pmatrix} 0.7519 & 0.7500 \\ 0.2481 & 0.2500 \end{pmatrix}$	$\begin{pmatrix} 0.75 & 0.75 \\ 0.25 & 0.25 \end{pmatrix}$
$N_{A_{kode}}$	$\begin{pmatrix} 0.2500 & 0.2481 \\ 0.7500 & 0.7519 \end{pmatrix}$	$\begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}$

Dan didapatkan bobot vektor dari matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-22.

Tabel 5.4-22 Tabel Nilai Bobot Vektor Uji Coba 51 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
UA_{Waktu}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{memori}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{jarak}	$(0.7509; 0.2491)^Z$	$(0.75; 0.25)^Z$
UA_{kode}	$(0.2491; 0.7509)^Z$	$(0.25; 0.75)^Z$

Didapatkan hasil nilai akhir untuk metode GA dan ACO berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-23.

Tabel 5.4-23 Tabel Nilai Akhir P Uji Coba 51 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
$P(GA)$	0.4785	0.4765
$P(ACO)$	0.5215	0.5235

5.4.6. Pembentukan Matriks Uji Coba 52 nodes

Hasil akhir pembentukan matriks turunan pada uji coba 52 nodes berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-24.

Tabel 5.4-24 Tabel *Matrix* Turunan Uji Coba 52 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
A_{waktu}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{memori}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{jarak}	$\begin{pmatrix} 1.0000 & 3.0000 \\ 0.3300 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 3 \\ 0.33 & 1 \end{pmatrix}$
A_{kode}	$\begin{pmatrix} 1.0000 & 0.3300 \\ 3.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}$

Dengan hasil normalisasi matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-25.

Tabel 5.4-25 Tabel Normalisasi *Matrix* Uji Coba 52 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
$N_{A_{waktu}}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{memori}}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{jarak}}$	$\begin{pmatrix} 0.7519 & 0.7500 \\ 0.2481 & 0.2500 \end{pmatrix}$	$\begin{pmatrix} 0.75 & 0.75 \\ 0.25 & 0.25 \end{pmatrix}$
$N_{A_{kode}}$	$\begin{pmatrix} 0.2500 & 0.2481 \\ 0.7500 & 0.7519 \end{pmatrix}$	$\begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}$

Dan didapatkan bobot vektor dari matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-26.

Tabel 5.4-26 Tabel Nilai Bobot Vektor Uji Coba 52 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
UA_{waktu}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{memori}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{jarak}	$(0.7509; 0.2491)^Z$	$(0.75; 0.25)^Z$
UA_{kode}	$(0.2491; 0.7509)^Z$	$(0.25; 0.75)^Z$

Didapatkan hasil nilai akhir untuk metode GA dan ACO berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-27.

Tabel 5.4-27 Tabel Nilai Akhir P Uji Coba 52 Nodes

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
$P(GA)$	0.4785	0.4765
$P(ACO)$	0.5215	0.5235

5.4.7. Pembentukan Matriks Uji Coba 76 nodes

Hasil akhir pembentukan matriks turunan pada uji coba 52 nodes berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-24.

Tabel 5.4-28

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
A_{waktu}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{memori}	$\begin{pmatrix} 1.0000 & 0.1000 \\ 9.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.1 \\ 9 & 1 \end{pmatrix}$
A_{jarak}	$\begin{pmatrix} 1.0000 & 0.3300 \\ 3.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}$
A_{kode}	$\begin{pmatrix} 1.0000 & 0.3300 \\ 3.0000 & 1.0000 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.33 \\ 3 & 1 \end{pmatrix}$

Dengan hasil normalisasi matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-25.

Tabel 5.4-29

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
$N_{A_{waktu}}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{memori}}$	$\begin{pmatrix} 0.1000 & 0.0909 \\ 0.9000 & 0.9091 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{pmatrix}$
$N_{A_{jarak}}$	$\begin{pmatrix} 0.2500 & 0.2481 \\ 0.7500 & 0.7519 \end{pmatrix}$	$\begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}$
$N_{A_{kode}}$	$\begin{pmatrix} 0.2500 & 0.2481 \\ 0.7500 & 0.7519 \end{pmatrix}$	$\begin{pmatrix} 0.25 & 0.25 \\ 0.75 & 0.75 \end{pmatrix}$

Dan didapatkan bobot vektor dari matriks turunan berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-26.

Tabel 5.4-30

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
UA_{waktu}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{memori}	$(0.0955; 0.9045)^Z$	$(0.1; 0.9)^Z$
UA_{jarak}	$(0.2481; 0.7519)^Z$	$(0.25; 0.75)^Z$
UA_{kode}	$(0.2481; 0.7519)^Z$	$(0.25; 0.75)^Z$

Didapatkan hasil nilai akhir untuk metode GA dan ACO berdasarkan perhitungan hasil komputasi program dan komputasi manual dapat dilihat pada Tabel 5.4-27.

Tabel 5.4-31

	<i>Perhitungan program</i>	<i>Perhitungan manual</i>
$P(GA)$	0.1904	0.1915
$P(ACO)$	0.8096	0.8085

5.5. Evaluasi Uji Coba

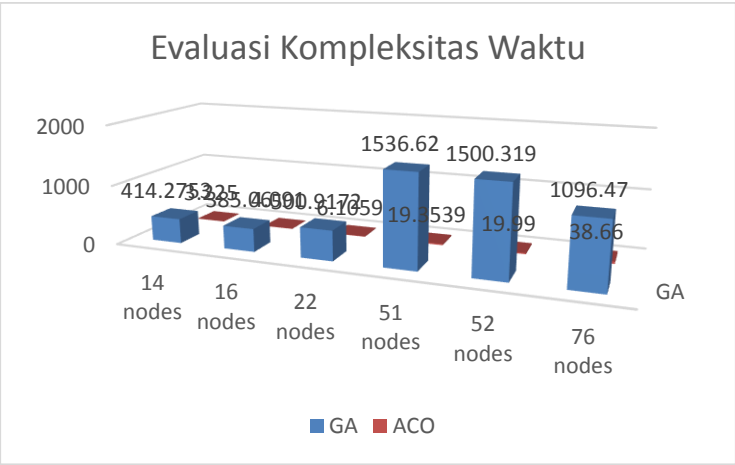
Pada subbab ini akan dilakukan evaluasi dari hasil semua uji coba yang sudah dilakukan.

5.5.1. Evaluasi Kompleksitas Waktu

Pada tahap ini kompleksitas waktu dari uji coba 14 nodes, 16 nodes, 22 nodes, 51 nodes, dan 52 nodes akan dievaluasi. Berdasarkan hasil uji coba pada subbab sebelumnya, secara keseluruhan kompleksitas waktu ditunjukkan pada Tabel 5.5-1 dan Gambar 5.5-1.

Tabel 5.5-1 Evaluasi Kompleksitas Waktu

Uji Coba	Hasil GA	Hasil ACO
14 nodes	414.2753s	3.225s
16 nodes	385.06s	4.091s
22 nodes	500.9172s	6.1059s
51 nodes	1536.62s	19.3539s
52 nodes	1500.319s	19.99s
76 nodes	1096.47s	20.6s



Gambar 5.5-1 Grafik Evaluasi Kompleksitas Waktu

Dari data diatas dapat diketahui kompleksitas waktu berbanding lurus dengan jumlah *nodes* pada permasalahan TSP. Semakin banyak *nodes* pada permasalahan TSP, maka semakin besar waktu yang dibutuhkan metode GA dan ACO untuk menyelesaikan permasalahan tersebut.

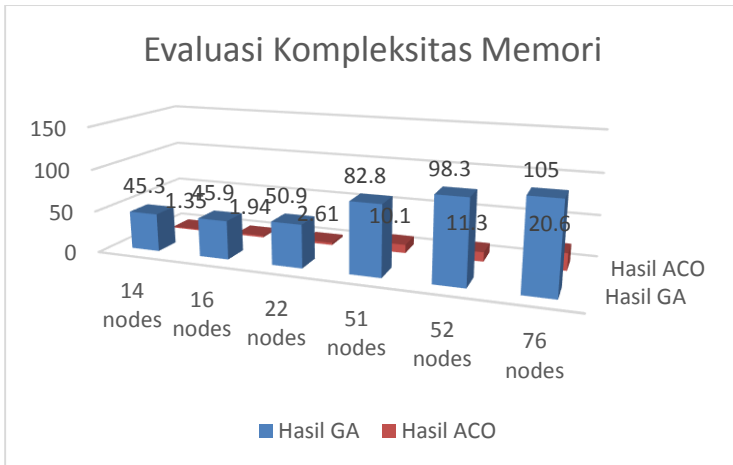
5.5.2. Evaluasi Kompleksitas Memori

Pada tahap ini kompleksitas memori dari uji coba 14 *nodes*, 16 *nodes*, 22 *nodes*, 51 *nodes*, dan 52 *nodes* akan dievaluasi. Berdasarkan hasil uji coba pada subbab sebelumnya, secara keseluruhan kompleksitas memori ditunjukkan pada Tabel 5.5-2 dan Gambar 5.5-2.

Tabel 5.5-2 Evaluasi Kompleksitas Memori

Uji Coba	Hasil GA	Hasil ACO
14 <i>nodes</i>	45.3 MB	1.35 MB
16 <i>nodes</i>	45.9 MB	1.94 MB
22 <i>nodes</i>	50.9 MB	2.61 MB
51 <i>nodes</i>	82.8 MB	10.1 MB
52 <i>nodes</i>	98.3 MB	11.3 MB
76 <i>nodes</i>	105 MB	20.6 MB

Dari data diatas dapat diketahui kompleksitas memori berbanding lurus dengan jumlah *nodes* pada permasalahan TSP. Semakin banyak *nodes* pada permasalahan TSP, maka semakin besar memoei yang dibutuhkan metode GA dan ACO untuk menyimpan penyelesaian permasalahan tersebut.



Gambar 5.5-2 Grafik Evaluasi Kompleksitas Memori

5.5.3. Evaluasi Akurasi Jarak Optimal

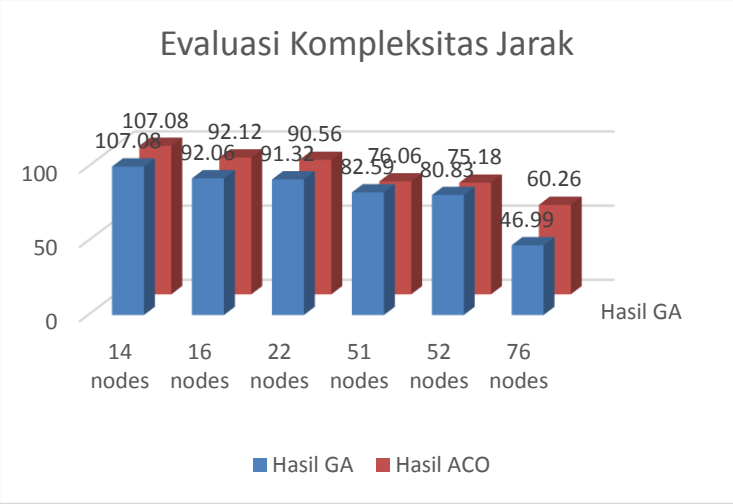
Pada tahap ini jarak optimal dari uji coba 14 *nodes*, 16 *nodes*, 22 *nodes*, 51 *nodes*, dan 52 *nodes* akan dievaluasi. Berdasarkan hasil uji coba pada subbab sebelumnya, secara keseluruhan jarak optimal ditunjukkan pada Tabel 5.5-3 dan Gambar 5.5-3.

Tabel 5.5-3 Evaluasi Akurasi Jarak Optimal

Uji Coba	Akurasi GA	Akurasi ACO
14 nodes	7.08% lebih baik	7.08% lebih baik
16 nodes	92.06%	92.12%
22 nodes	91.32%	90.56%
51 nodes	82.59%	76.06%
52 nodes	80.83%	75.18%
76 nodes	47%	60.26%

Dari data diatas dapat diketahui akurasi kompleksitas jarak optimal berbanding terbalik dengan jumlah *nodes* pada permasalahan TSP. Semakin banyak *nodes* pada

permasalahan TSP, maka tingkat akurasi dari pengerjaan metode GA dan ACO untuk penyelesaian permasalahan TSP semakin menurun.



Gambar 5.5-3 Grafik Evaluasi Kompleksitas Jarak

5.5.4. Evaluasi Akhir Pairwise Comparison

Pada tahap ini nilai *P* uji coba 14 nodes, 16 nodes, 22 nodes, 51 nodes, dan 52 nodes akan dievaluasi. Berdasarkan hasil uji coba pada subbab sebelumnya, secara keseluruhan nilai *P* ditunjukkan pada Tabel 5.5-4 dan Gambar 5.5-1.

Tabel 5.5-4 Evaluasi Nilai *P*

Uji Coba	Nilai <i>P</i> GA	Nilai <i>P</i> ACO
14 nodes	0.334	0.666
16 nodes	0.334	0.666
22 nodes	0.4765	0.5235
51 nodes	0.4765	0.5235
52 nodes	0.4765	0.5235
76 nodes	0.1915	0.8085

Dari data diatas didapat rata-rata dari nilai P metode GA adalah 0.6185, dan rata-rata dari nilai P metode ACO adalah 0.3815. Dapat disimpulkan metode ACO lebih baik dari metode GA dalam penyelesaian masalah TSP 14 *nodes*, 16 *nodes*, 22 *nodes*, 51 *nodes*, dan 52 *nodes*. Kesimpulan tersebut didasarkan nilai P yang diperoleh metode ACO lebih besar dari nilai P yang diperoleh metode GA. Secara keleruhan metode ACO lebih baik dari metode GA.

(halaman ini sengaja dikosongkan)

BAB VI

KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan perangkat lunak lebih lanjut.

6.1. Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap pembuatan program pembandingan kinerja *varians* dari dua metode kecerdasan buatan yang telah ditentukan untuk proses penyelesaian masalah TSP dapat diambil kesimpulan sebagai berikut:

1. Metode GA unggul pada kriteria hasil jarak optimal, dengan batasan *nodes* pada permasalahan tidak lebih dari 52 *nodes*. Sementara metode ACO unggul pada tiga kriteria lainnya yaitu kompleksitas waktu, kompleksitas memori, dan tingkat kesulitan dalam implementasi.
2. Dari proses *Pairwise Comparison* diketahui metode ACO lebih baik dengan nilai P rata-rata 0.6185, sedangkan GA memiliki nilai P rata-rata 0.3815.

6.2. Saran

Saran yang diberikan untuk pengembangan aplikasi pembandingan kinerja *varians* dari dua metode kecerdasan buatan yang telah ditentukan untuk proses penyelesaian masalah TSP pada Tugas Akhir adalah peninjauan ulang kembali dalam menentukan skala derajat kepentingan pada proses perbandingan *Pairwise Comparison*.

LAMPIRAN

Lampiran A 1 Data Masukan *burma14*

Nomor Kota	Sumbu X	Sumbu Y
1	1647	9610
2	1647	9444
3	2009	9254
4	2239	9337
5	2523	9724
6	2200	9605
7	2047	9702
8	1720	9629
9	1630	9738
10	1405	9812
11	1653	9738
12	2152	9559
13	1941	9713
14	2009	9455

Lampiran A 2 Data Masukan *ulysses16*

Nomor Kota	Sumbu X	Sumbu Y
1	3824	2042
2	3957	2615
3	4056	2532
4	3626	2312
5	3348	1054
6	3756	1219
7	3842	1311
8	3752	2044
9	4123	910
10	4117	1305
11	3608	-521
12	3847	1513
13	3815	1535
14	3751	1517
15	3549	1432
16	3936	1956

Lampiran A 3Data Masukan *ulysses22*

Nomor Kota	Sumbu X	Sumbu Y
1	3824	2042
2	3957	2615
3	4056	2532
4	3626	2312
5	3348	1054
6	3756	1219
7	3842	1311
8	3752	2044
9	4123	910
10	4117	1305
11	3608	-521
12	3847	1513
13	3815	1535
14	3751	1517
15	3549	1432
16	3936	1956
17	3809	2436
18	3609	2300
19	4044	1357
20	4033	1415
21	4037	1423
22	3757	2256

Lampiran A 4 Data Masukan *att48* (Bagian 1)

Nomor Kota	Sumbu X	Sumbu Y
1	6734	1453
2	2233	10
3	5530	1424
4	401	841
5	3082	1644
6	7608	4458
7	7573	3716
8	7265	1268
9	6898	1885
10	1112	2049
11	5468	2606
12	5989	2873
13	4706	2674
14	4612	2035
15	6347	2683
16	6107	669
17	7611	5184
18	7462	3590
19	7732	4723
20	5900	3561
21	4483	3369
22	6101	1110
23	5199	2182
24	1633	2809
25	4307	2322

Lampiran A 5 Data Masukan *att48* (Bagian 2)

Nomor Kota	Sumbu X	Sumbu Y
26	675	1006
27	7555	4819
28	7541	3981
29	3177	756
30	7352	4506
31	7545	2801
32	3245	3305
33	6426	3173
34	4608	1198
35	23	2216
36	7248	3779
37	7762	4595
38	7392	2244
39	3484	2829
40	6271	2135
41	4985	140
42	1916	1569
43	7280	4899
44	7509	3239
45	10	2676
46	6807	2993
47	5185	3258
48	3023	1942

Lampiran A 6 Data Masukan *berlin52* (Bagian 1)

Nomor Kota	Sumbu X	Sumbu Y
1	565	575
2	25	185
3	345	750
4	945	685
5	845	655
6	880	660
7	25	230
8	525	1000
9	580	1175
10	650	1130
11	1605	620
12	1220	580
13	1465	200
14	1530	5
15	845	680
16	725	370
17	145	665
18	415	635
19	510	875
20	560	365
21	300	465
22	520	585
23	480	415
24	835	625
25	975	580
26	1215	245

Lampiran A 7 Data Masukan *berlin52* (Bagian 2)

Nomor Kota	Sumbu X	Sumbu Y
27	1320	315
28	1250	400
29	660	180
30	410	250
31	420	555
32	575	665
33	1150	1160
34	700	580
35	685	595
36	685	610
37	770	610
38	795	645
39	720	635
40	760	650
41	475	960
42	95	260
43	875	920
44	700	500
45	555	815
46	830	485
47	1170	65
48	830	610
49	605	625
50	595	360
51	1340	725
52	1740	245

Lampiran A 8 Data Masukan *eil76* (Bagian 1)

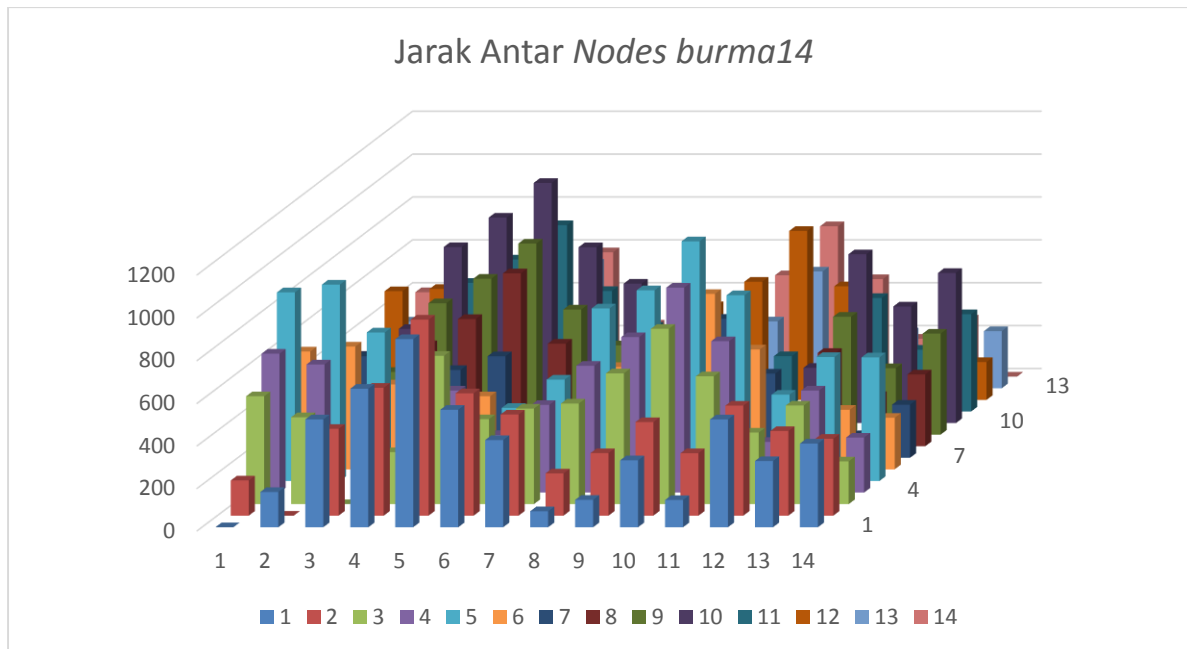
Nomor Kota	Sumbu X	Sumbu Y
1	22	22
2	36	26
3	21	45
4	45	35
5	55	20
6	33	34
7	50	50
8	55	45
9	26	53
10	40	66
11	55	65
12	35	51
13	62	35
14	62	57
15	62	24
16	21	36
17	33	44
18	3	56
19	62	48
20	66	14
21	44	13
22	26	13
23	11	28
24	7	43
25	17	64
26	41	46

Lampiran A 9 Data Masukan *eil76* (Bagian 2)

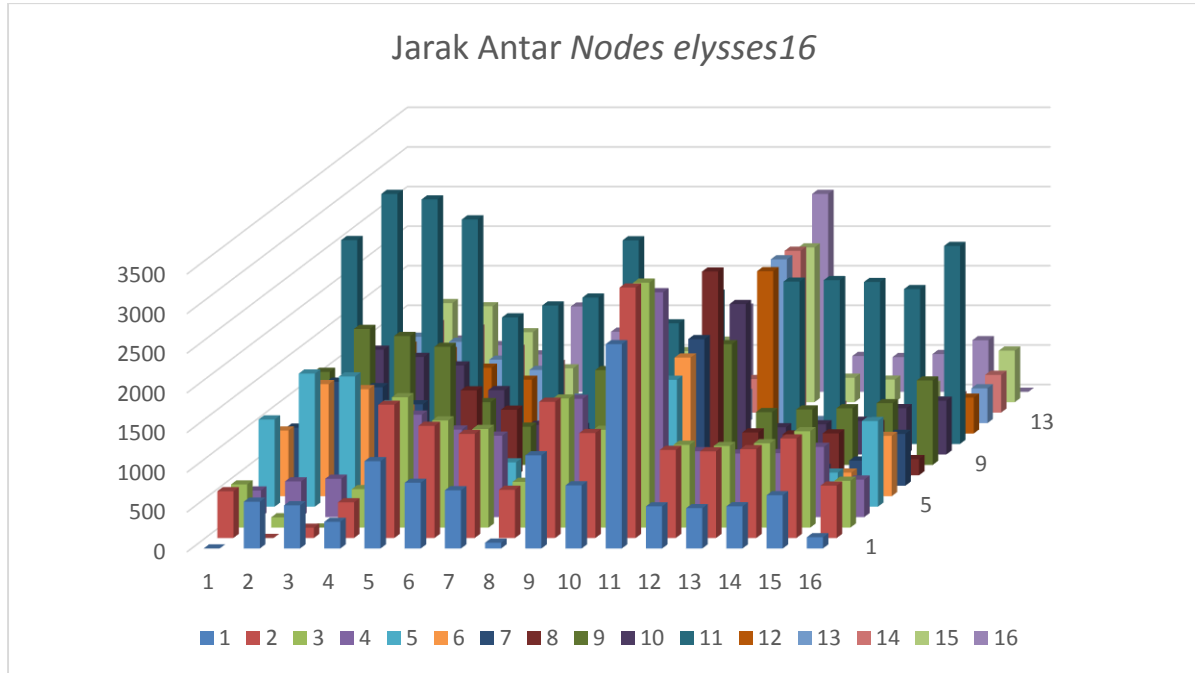
Nomor Kota	Sumbu X	Sumbu Y
27	55	34
28	35	16
29	52	26
30	43	26
31	31	76
32	22	53
33	26	23
34	50	40
35	55	50
36	54	10
37	60	15
38	47	66
39	30	60
40	30	50
41	12	17
42	15	14
43	16	13
44	21	48
45	50	30
46	51	42
47	50	15
48	48	21
49	12	38
50	15	56
51	23	33
52	54	38
53	55	57

Lampiran A 10 Data Masukan *eil76* (Bagian 3)

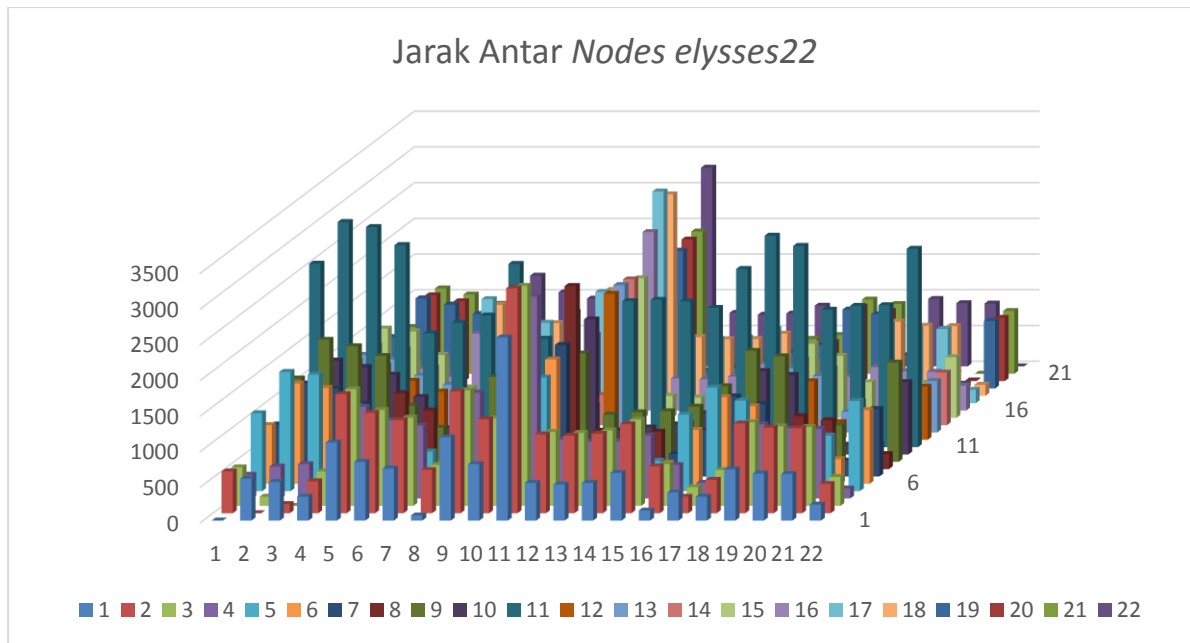
Nomor Kota	Sumbu X	Sumbu Y
54	67	41
55	10	70
56	6	25
57	65	27
58	40	60
59	70	64
60	64	4
61	36	6
62	30	20
63	20	30
64	15	5
65	50	70
66	57	72
67	45	42
68	38	33
69	50	4
70	66	8
71	53	5
72	35	60
73	27	24
74	40	20
75	40	37
76	40	40



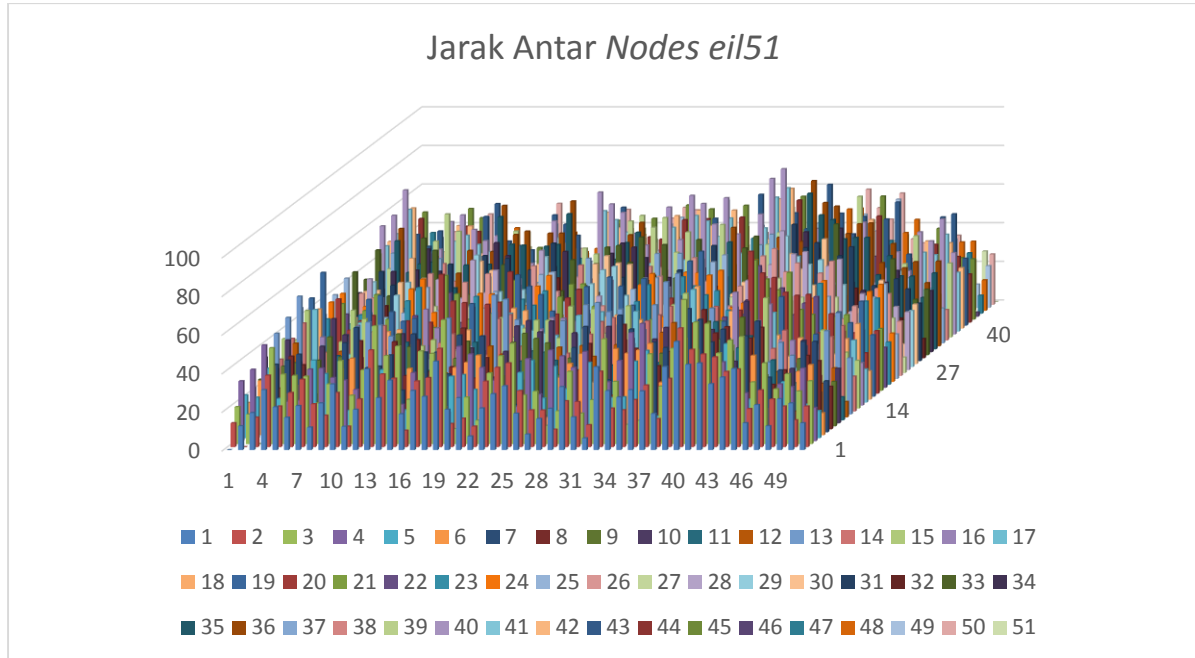
Gambar A 1 Grafik Jarak Antar Nodes *burma14*



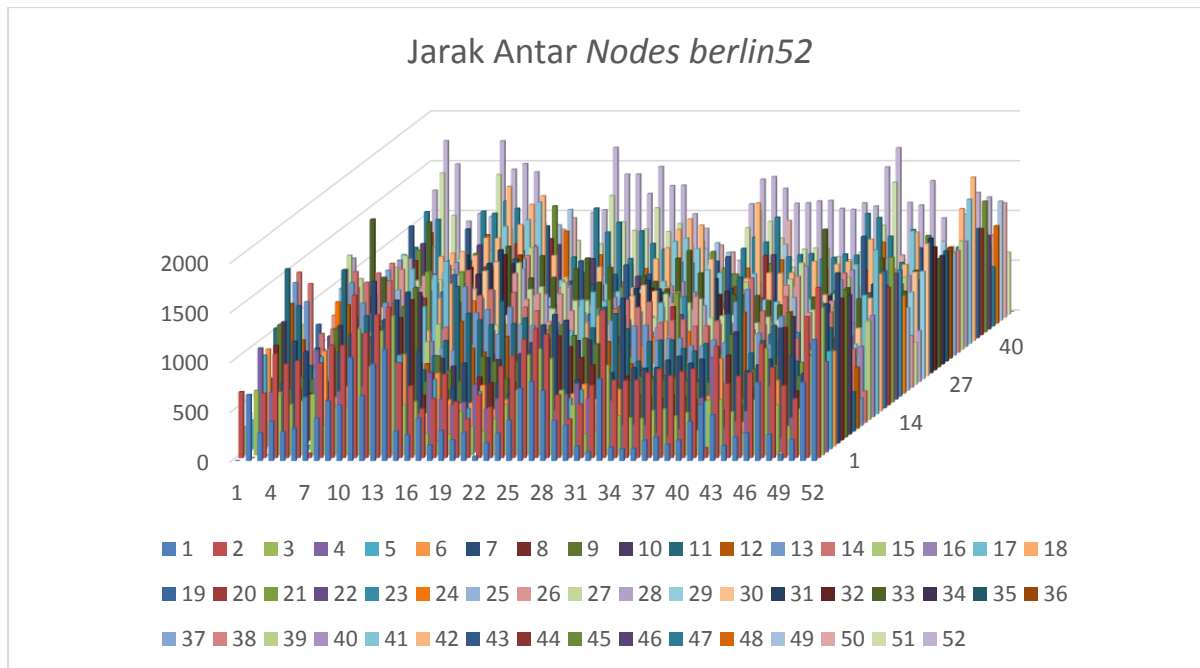
Gambar A 2 Grafik Jarak Antar Nodes elykses16



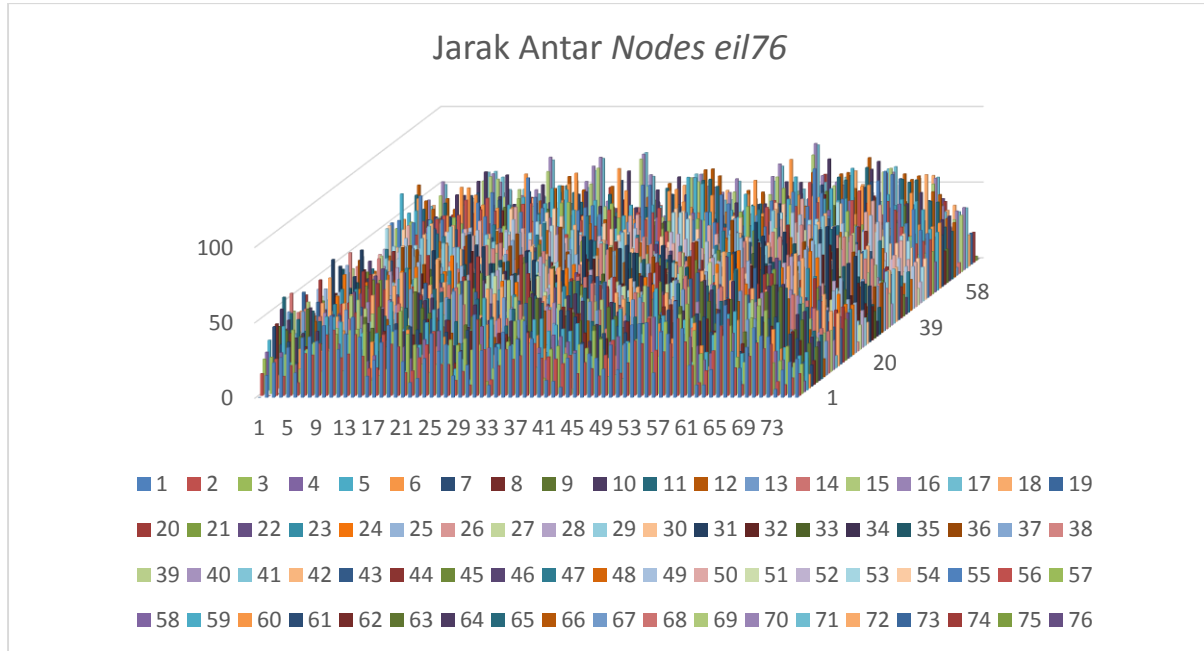
Gambar A 3 Grafik Jarak Antar *Nodes elykses22*



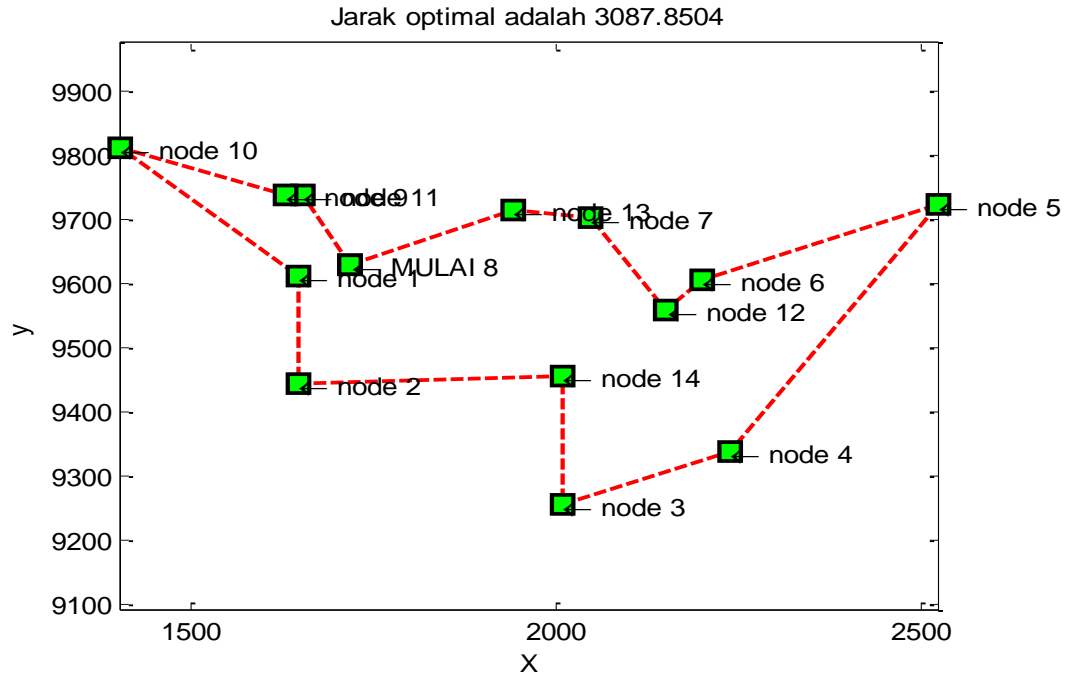
Gambar A 4 Grafik Jarak Antar Nodes *eil51*



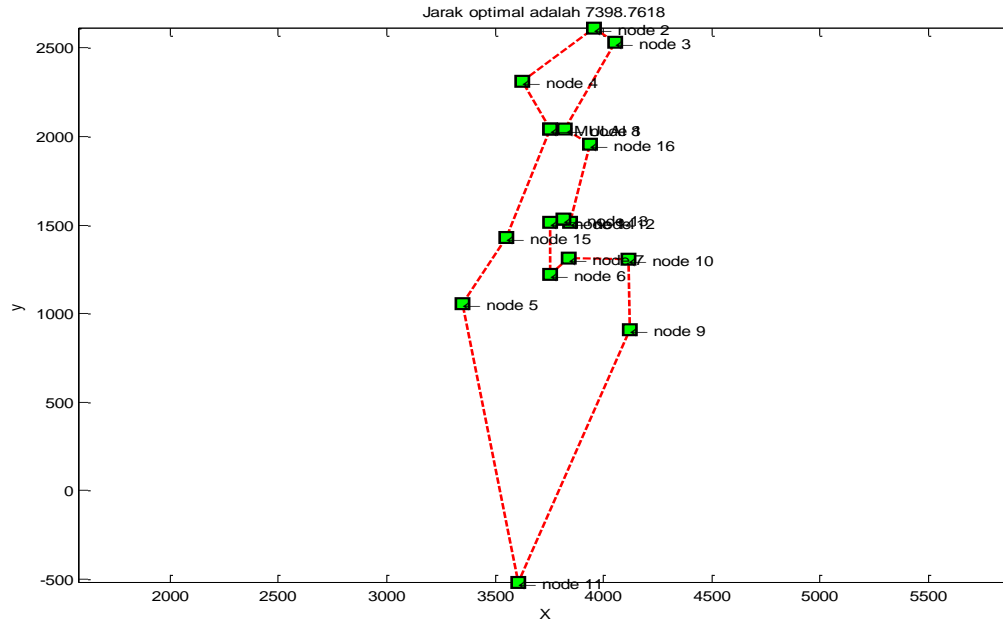
Gambar A 5 Grafik Jarak Antar Nodes *berlin52*



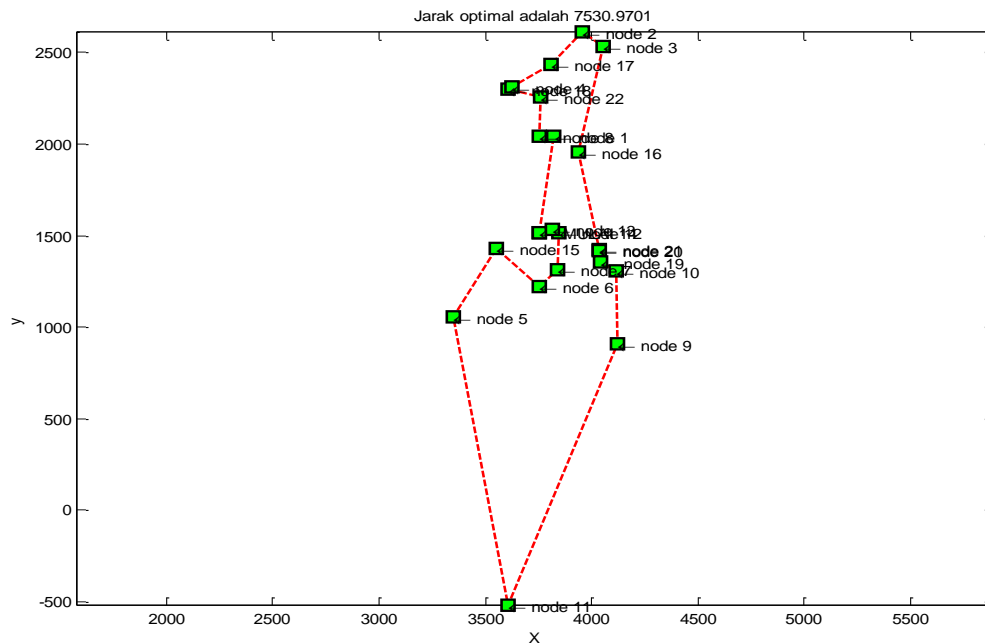
Gambar A 6 Grafik Jarak Antar *Nodes eil76*



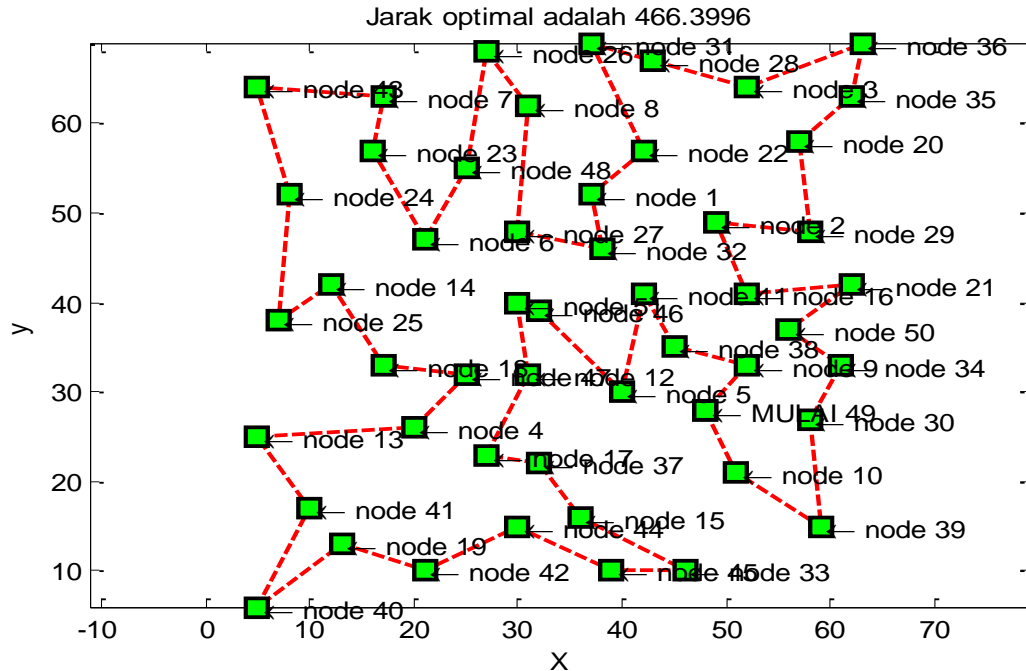
Gambar A 7 Gambar Hasil Terbaik Uji Coba 14 Nodes dengan Metode GA



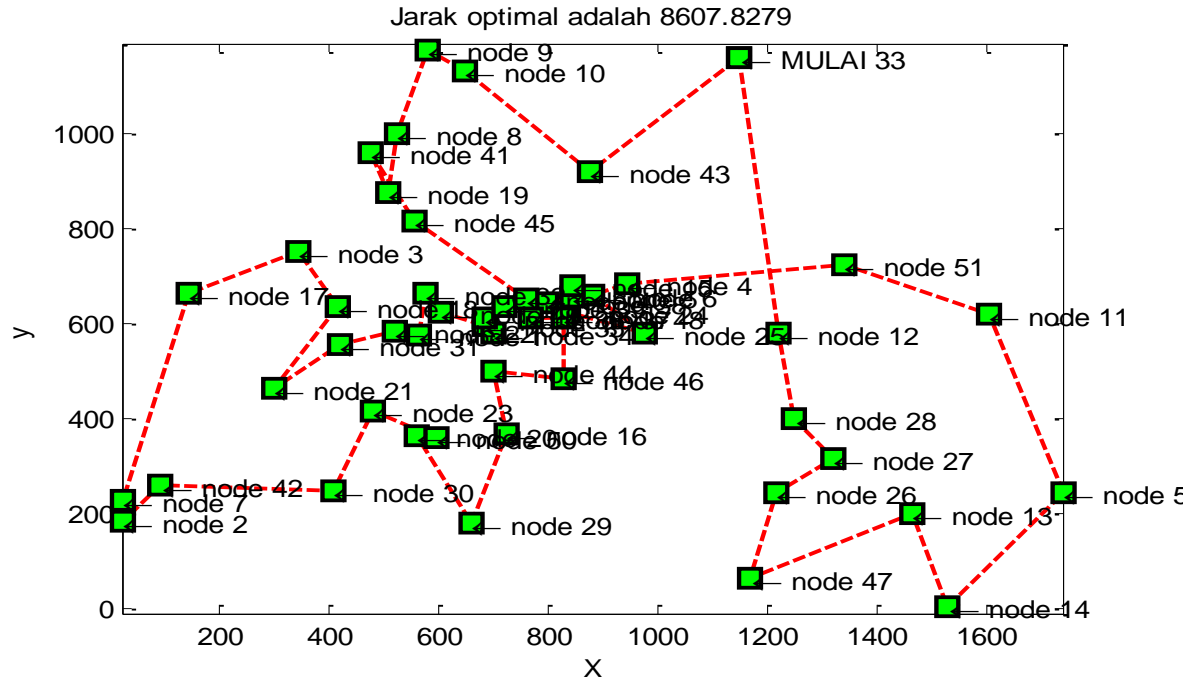
Gambar A 8 Gambar Hasil Terbaik Uji Coba 16 Nodes dengan Metode GA



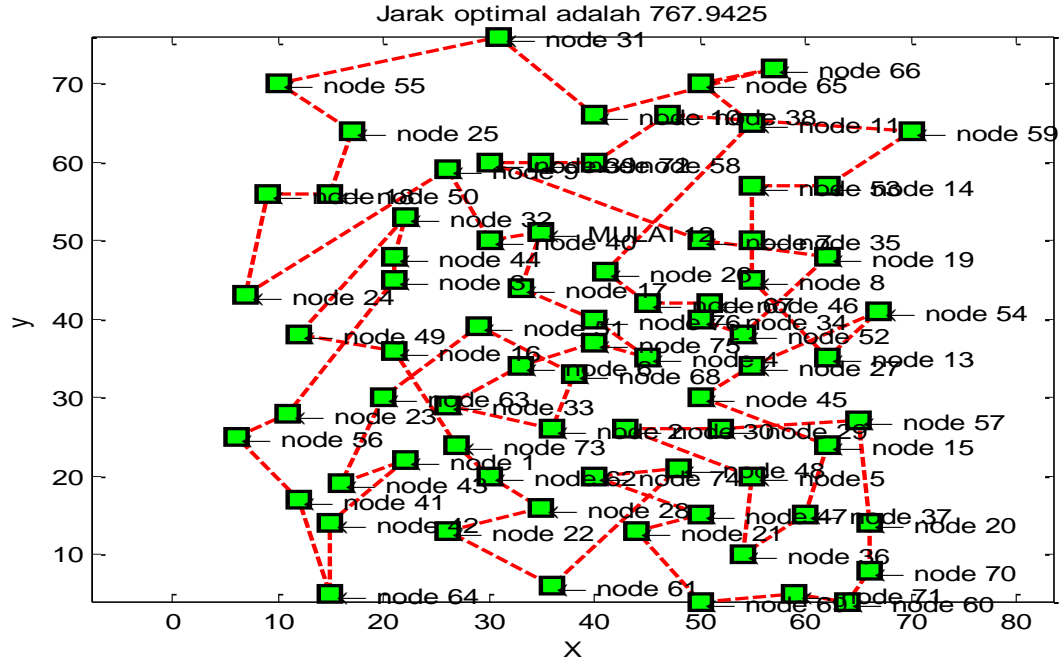
Gambar A 9 Gambar Hasil Terbaik Uji Coba 22 Nodes dengan Metode GA



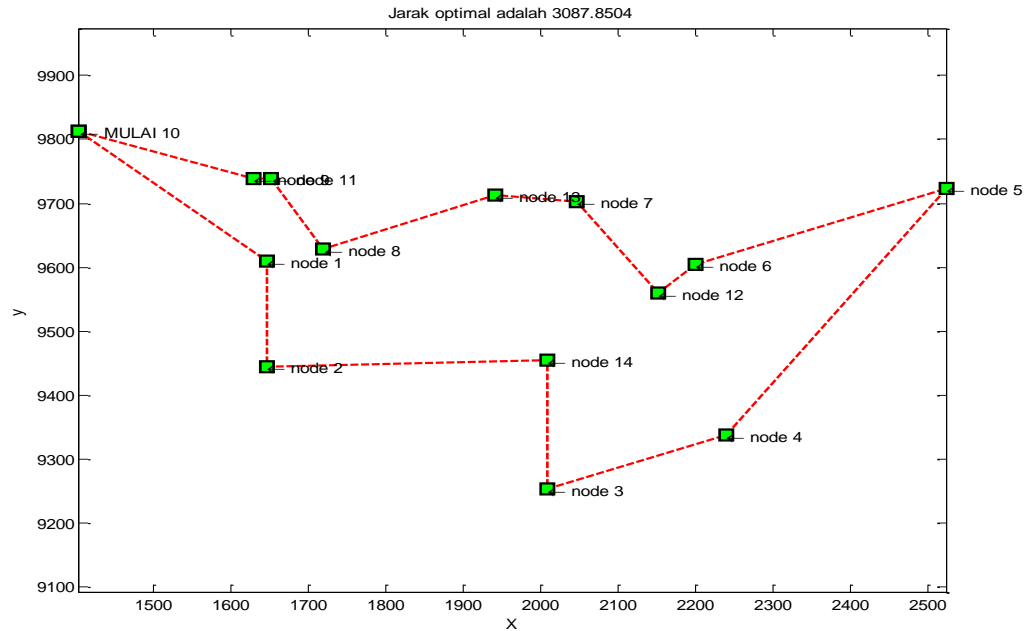
Gambar A 10 Gambar Hasil Terbaik Uji Coba 51 Nodes dengan Metode GA



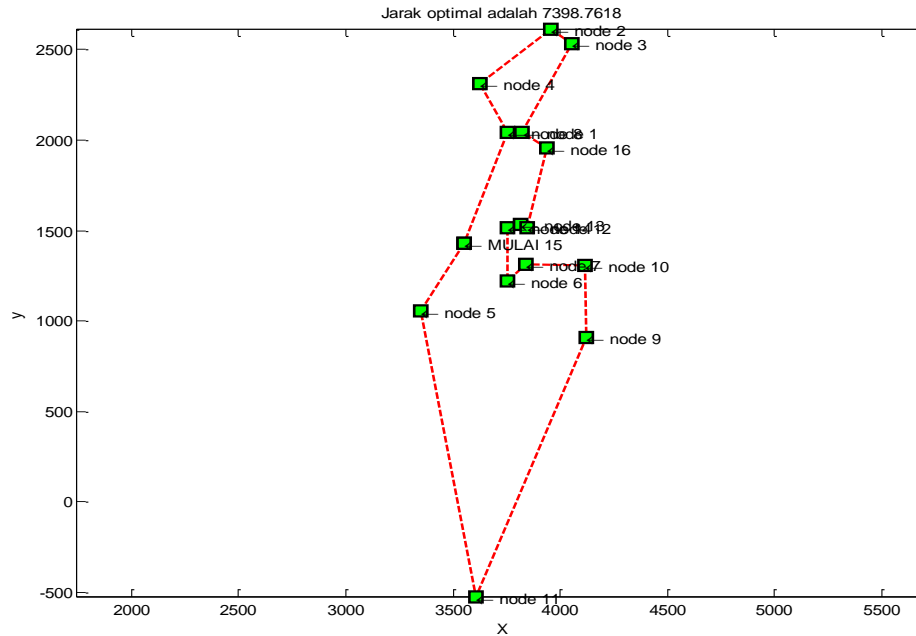
Gambar A 11 Gambar Hasil Terbaik Uji Coba 52 Nodes dengan Metode GA



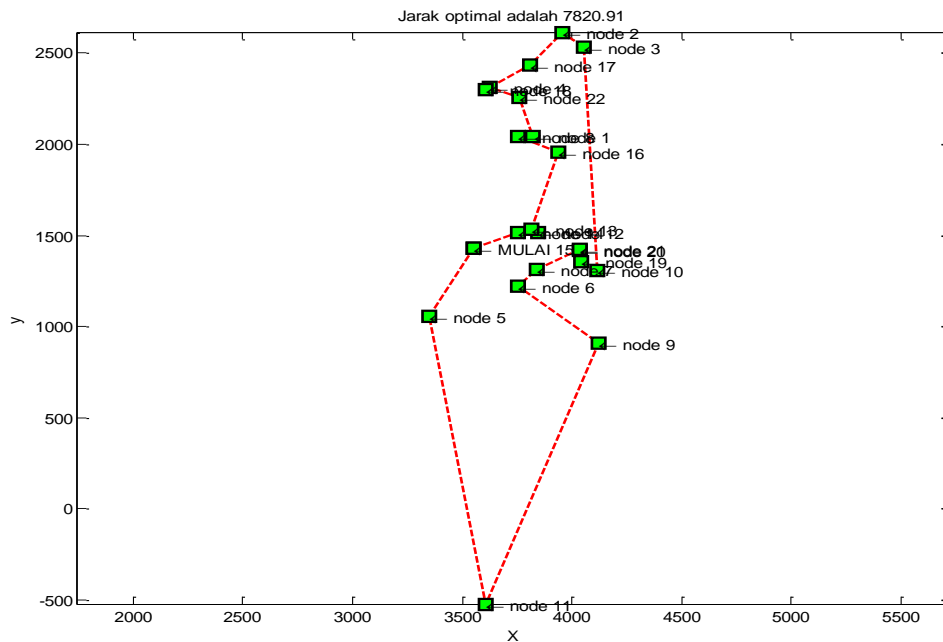
Gambar A 12 Gambar Hasil Terbaik Uji Coba 76 Nodes dengan Metode GA



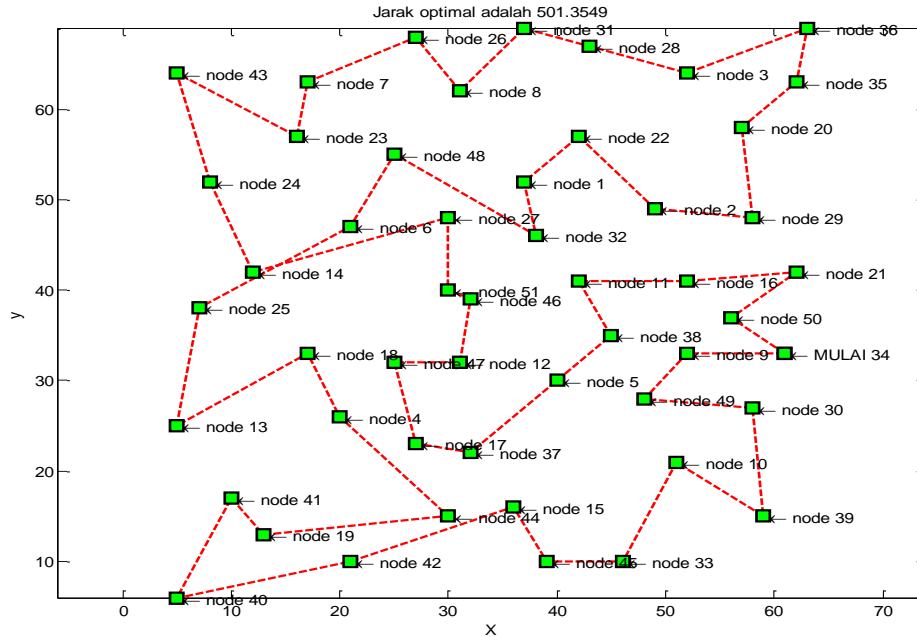
Gambar A 13 Gambar Hasil Terbaik Uji Coba 14 *Nodes* dengan Metode ACO



Gambar A 14 Gambar Hasil Terbaik Uji Coba 16 *Nodes* dengan Metode ACO

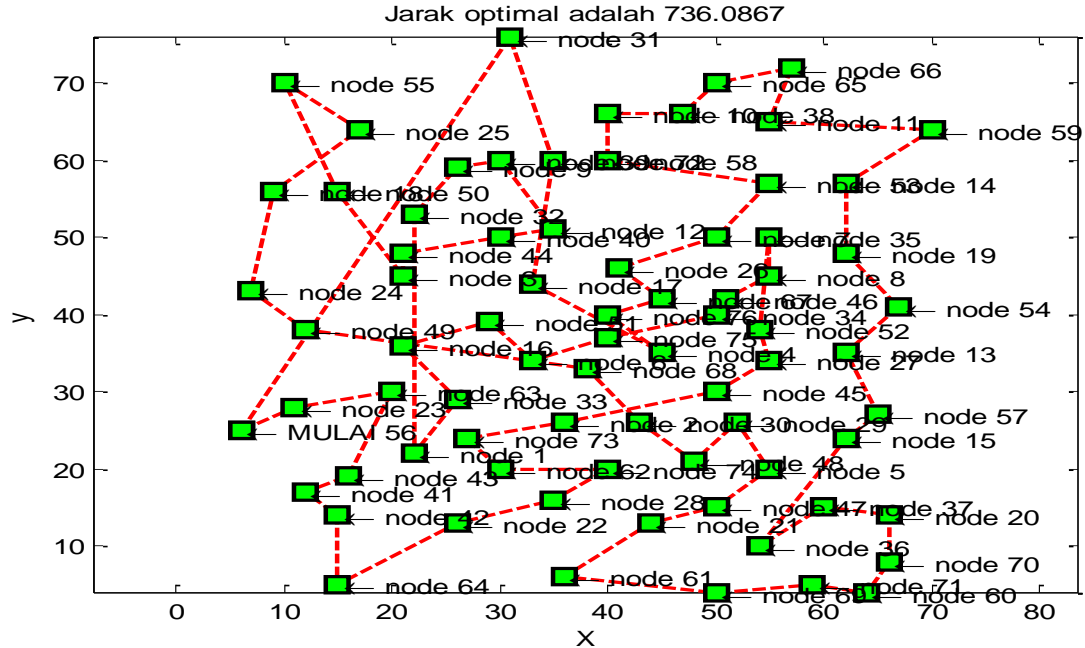


Gambar A 15 Gambar Hasil Terbaik Uji Coba 22 Nodes dengan Metode ACO



Gambar A 16 Gambar Hasil Terbaik Uji Coba 51 Nodes dengan Metode ACO

Gambar A 17 Gambar Hasil Terbaik Uji Coba 52 Nodes dengan Metode ACO



Gambar A 18 Gambar Hasil Terbaik Uji Coba 76 Nodes dengan Metode ACO

U1 =		
	0.0955	0.9045
U2 =		
	0.0955	0.9045
U3 =		
	0.5000	0.5000
U4 =		
	0.1667	0.8333
PGA =		
	0.3308	
PACO =		
	0.6692	
ACO LEBIH BAIK		

Gambar A 19 Gambar Hasil *Pairwise Comparison* 14 Nodes

U1 =		
	0.0955	0.9045
U2 =		
	0.0955	0.9045
U3 =		
	0.5000	0.5000
U4 =		
	0.1667	0.8333
PGA =		
	0.3308	
PACO =		
	0.6692	
ACO LEBIH BAIK		

Gambar A 20 Gambar Hasil *Pairwise Comparison* 16 Nodes

U1 =		1
	0.0955	0.9045
U2 =		
	0.0955	0.9045
U3 =		
	0.7509	0.2491
U4 =		
	0.1667	0.8333
PGA =		
	0.4748	
FACO =		
	0.5252	
ACO LEBIH BAIK		

Gambar A 21 Gambar Hasil *Pairwise Comparison* 22 Nodes

U1 =		1
	0.0955	0.9045
U2 =		
	0.0955	0.9045
U3 =		
	0.7509	0.2491
U4 =		
	0.1667	0.8333
PGA =		
	0.4748	
FACO =		
	0.5252	
ACO LEBIH BAIK		

Gambar A 22 Gambar Hasil *Pairwise Comparison* 51 Nodes

U1 =		1
	0.0955	0.9045
U2 =		
	0.0955	0.9045
U3 =		
	0.7509	0.2491
U4 =		
	0.1667	0.8333
PGA =		
	0.4748	
PACO =		
	0.5252	
ACO LEBIH BAIK		

Gambar A 23 Gambar Hasil *Pairwise Comparison* 52 Nodes

U1 =		
	0.0955	0.9045
U2 =		
	0.0955	0.9045
U3 =		
	0.2491	0.7509
U4 =		
	0.2491	0.7509
PGA =		
	0.1904	
PACO =		
	0.8096	
ACO LEBIH BAIK		

Gambar A 24 Gambar Hasil *Pairwise Comparison* 76 Nodes

BIODATA PENULIS



Muhammad Ibrahim Oswaldo, lahir di Bandar Lampung pada tanggal 29 Agustus 1993. Penulis menempuh pendidikan mulai dari SD Islam Al-Hasanah 1999 – 2003, SD Islam Al-Azhar 08 Kembangan 2003 – 2005, SLTP Islam Al-Azhar 10 Kembangan 2005 – 2008, International Islamic Boarding School 2008 – 2010, dan S1 Teknik Informatika ITS 2010 – 2014. Bidang Studi yang diambil oleh penulis pada saat

kuliah di Teknik Informatika ITS adalah Komputasi Cerdas dan Visualisasi (KCV). Penulis dapat dihubungi melalui *e-mail*: ibamoswaldo@gmail.com